Simulation framework for the DG

ESS Detector Group Jamboree, DTU Risø, 2016-09-05

Thomas Kittelmann, ESS

Note: framework support & features not just due to me.
In particular Kelly & Xiao Xiao contribute heavily.

EUROPEAN SPALLATION SOURCE

What is our framework?

Technically it is:

- A code repository (dgcode)
- with a build system (dgbuild)
- with conventions of how to add code
- with possibilities to provide code to each other
- with associated issue tracker, wiki, etc.
- most content concerns Geant4 simulations

It is also:

- A way to collaborate on code projects
- Sharing expertise and tools as we go along
- A way to keep code “alive” and accessible
- A way to increase the “bus-factor” of projects

McStas, Vitess, MCNP are *applications*, Geant4 is a *toolkit*

Framework reference:
Kittelmann, et al. CHEP 2013.
doi:10.1088/1742-6596/513/5/052022

Outline of this presentation

- Will skip most details of how to install, how to use mercurial, how to put code where in dgcode, how to use dgbuild
  - Details are on the wiki
  - Sara will show some of this in her presentation
- Instead, will briefly showcase:
  - How simulation projects are put together
  - Available utilities for dealing with different aspects of a sim project
  - Some recent developments and additions
    - Won't go much into specific projects (many dedicated talks)
    - Some new features have dedicated talks: MCPL, NCrystal
- Due to time constraints, some slides are mainly here for reference and I will skip over them very fast!

Slides marked with this icon contain reference material – we won't go into everything today

The basics

Getting started, geometry, sim-script, visualisation

Ready to rumble?

Step 1.
Install base dependencies following platform-specific instructions on HowToInstallComputingPrereqs

Step 2.
Edit mercurial settings in the file ~/.hgrc (as per wiki instructions).
$> hg clone dg_dgcode

Step 3.
$> cd dg_dgcode
$> . bootstrap.sh
$> dgbuild

```
Package G4XSectDump done
  Creating python module GriffAnaUtils._init
Package GriffAnaUtils done
All done
dgbuild:
dgbuild: Successfully built and installed all enabled packages!
dgbuild:
dgbuild: Summary:
dgbuild:   Installation directory           : /home/thki/work/repos/tkdgcode/install
dgbuild:   System                           : Linux-3.19.3-200.fc21.x86_64
dgbuild:   User configuration variables[*]  : ONLY='Framework/*'
dgbuild:   Required dependencies            : Boost[1.55.0] C[GNU/4.9.2] CMake[3.0.2] CXX[GNU/4.9.2]
dgbuild:                                      Python[2.7.8]
dgbuild:   Optional dependencies present    : Fortran[GNU/4.9.2] Garfield[unknown] Geant4[10.0.3]
dgbuild:                                      HDF5[1.8.13] OSG[3.2.1] ROOT[5.34/28] ZLib[1.2.8]
dgbuild:   Optional dependencies missing[*] : <none>
dgbuild:   50 packages built successfully   : Core DMSCUtils DSFMT DevTools EvtFile G4CollectFilters
dgbuild:                                      G4CustomPyGen G4DataCollect G4FastHPModel
dgbuild:                                      G4HadronProcessFHP G4Interfaces G4Launcher
dgbuild:                                      G4MCNPGun G4Materials G4McStas G4NeutronAceTS
dgbuild:                                      G4OSG G4PhysListsFHP_ACE G4PhysListsTS G4PhysicsLists
dgbuild:                                      ... (30 more, supply --verbose to see all)
dgbuild:   99 packages skipped due to [*]   : B10MathUtils BandGem BandGemBug148 BasicExamples
dgbuild:                                      CoreTests CrystalPerfPlots EBTCScripts FortranExamples
dgbuild:                                      G4B10Common G4Examples G4GeoAceTS G4GeoB10Film
dgbuild:                                      G4GeoBandGem G4GeoBandGemBug148 G4GeoEBTC
dgbuild:                                      G4GeoHe3Tubes G4GeoLoki G4GeoReactorSim G4GeoRecover
dgbuild:                                      G4GeoScint ... (79 more, supply --verbose
dgbuild:                                      to see all)
dgbuild:
dgbuild: Note that only Framework/ packages were enabled by default:
dgbuild:
dgbuild:    - To enable pkgs for a given project do: dgbuild -p<projectname>
dgbuild:    - To enable all pkgs do: dgbuild -a
dgbuild:
dgbuild: You are all set to begin using the software!
dgbuild:
dgbuild: To see available applications, type "ess_" and hit the TAB key twice.
dgbuild:
(thki@tklenovo2014 tkdgcode)>
```

Optional step 4.
Install other dependencies as desired, perhaps using dgcode utilities, such as:
ess_devtools_installgeant4

Only Step 3. needs repeating during daily work

Relevant wiki pages:
Mercurial
CodingFramework
HowToInstallComputingPrereqs

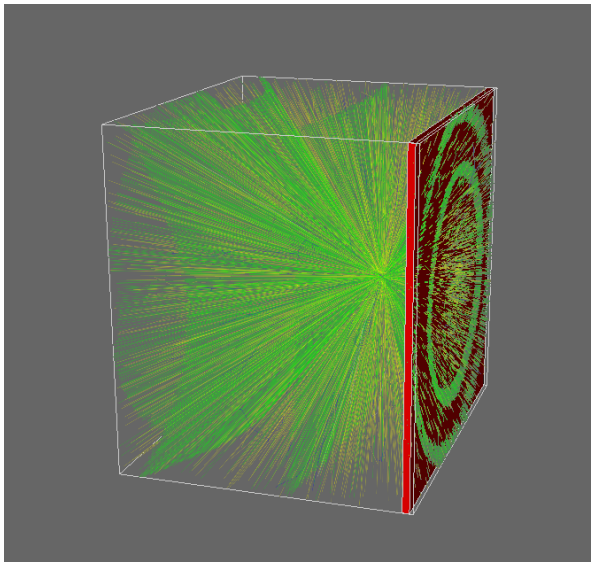
Create skeleton code for new sim projects

(because people anyway starts by copy+edit of existing examples)

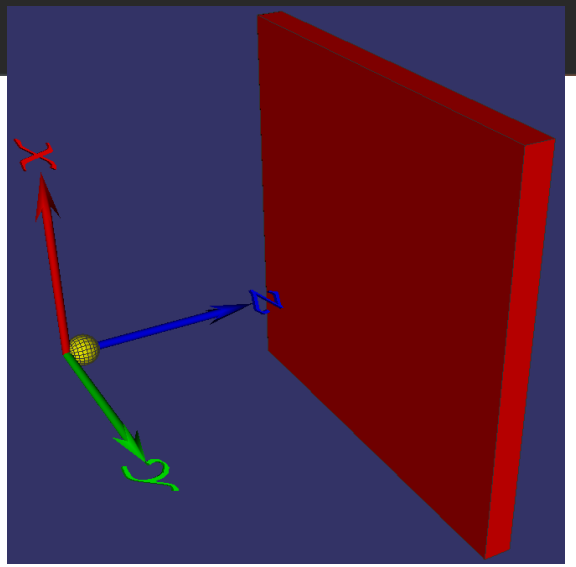
https://ess-ics.atlassian.net/wiki/display/DG/How+to+start+a+new+simulation+project

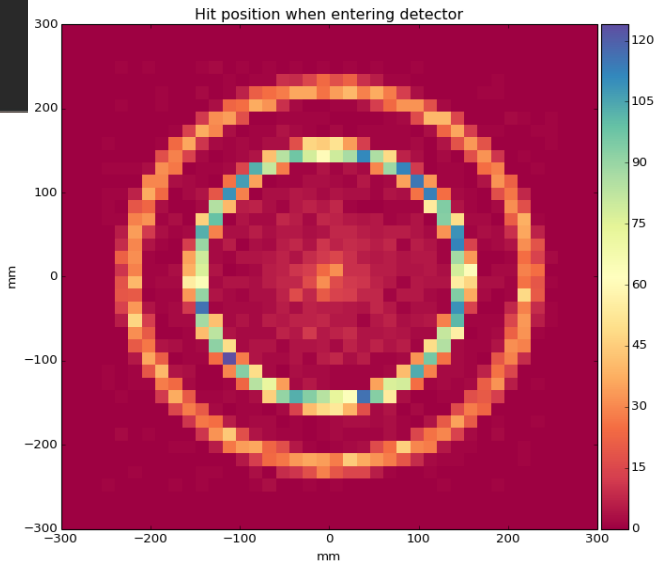
```
(thki@tklenovo2014 tkdgcode)> ess_devtools_newsimproject TriCorder
Created file: Projects/TriCorder/G4GeoTriCorder/pkg.info
Created file: Projects/TriCorder/G4GeoTriCorder/pycpp_GeoTriCorder/geometry_module.cc
Created file: Projects/TriCorder/TriCorder/pkg.info
Created file: Projects/TriCorder/TriCorder/scripts/simanachain
Created file: Projects/TriCorder/TriCorder/scripts/scanana
Created file: Projects/TriCorder/TriCorder/scripts/scan
Created file: Projects/TriCorder/TriCorder/scripts/test
Created file: Projects/TriCorder/TriCorder/scripts/sim
Created file: Projects/TriCorder/TriCorder/app_ana/analysis_program.cc

Created 9 new files from the Examples/Skeletons/SkeletonSP skeleton under:

        Projects/TriCorder

Now you can go through them and replace their contents as needed for your project.

Do not forget to update documentation in comments and pkg.info files and make sure
that everything is tested with at least "dgbuild --project=TriCorder -dt" before
committing anything to the repository!

(thki@tklenovo2014 tkdgcode)>
```

--> ess_tricorder_sim



In reality, just 2-3 files needs to be modified to get started with a new project.





The simulation script (“sim-script”)

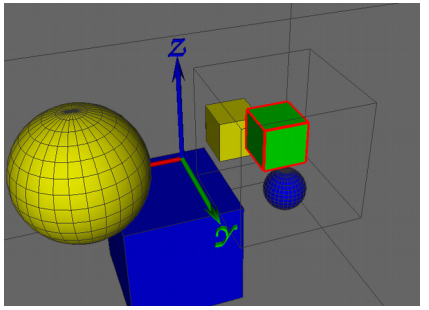
Tricorder/scripts/sim → can run as ess_tricorder_sim

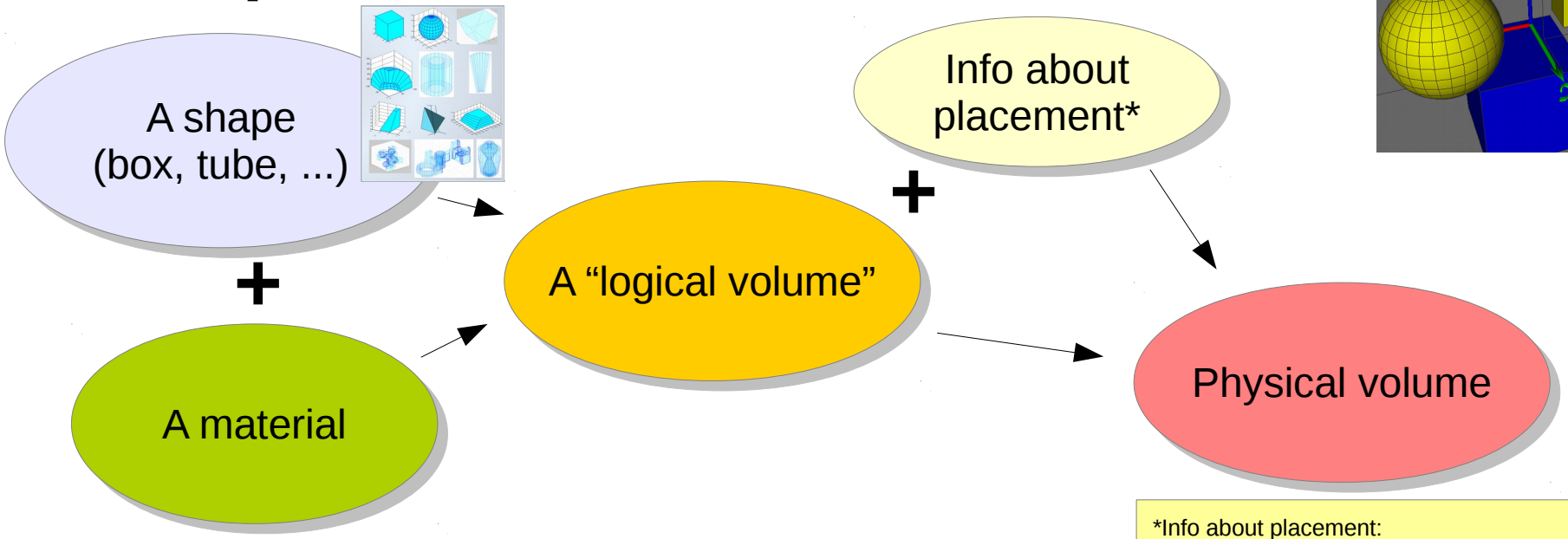
geometry module + generator module = simulation application

```
#!/usr/bin/env python

import G4GeoTriCorder.GeoTriCorder as geomodule
geo = geomodule.create()
geo.sample_posz_mm = 5.0

import G4StdGenerators.SimpleGen as genmodule
gen = genmodule.create()
gen.particleName = 'neutron'
gen.neutron_wavelength_aangstrom = 2.2

import G4Launcher
launcher = G4Launcher(geo,gen)
launcher.setOutput('tricorder','REDUCED')#Gr
launcher.go()
```

```
(thki@tklenovo2014 tkdgcode)> ess_tricorder_sim -h
Usage: ess_tricorder_sim [options] [par1=val1] [par2=val2] [...]

This script allows you to simulate or visualise particles from the
G4StdGenerators/SimpleGen generator hitting the G4GeoTriCorder/GeoTriCorder
geometry. Note that in addition to the options below, you can override
parameters of the generator and geometry by supplying them on the commandline
like par=val. Furtermore note that as a special case, you can disable the
parameter validation by setting forcepars=yes.

Options:
  -h, --help            show this help message and exit
  -d, --dump            Dump parameters of both geometry and generator
  -g                    Dump parameters of just geometry
  -p                    Dump parameters of just generator
  -x                    Dump used cross-sections in text files
  --viewer              Experimental OpenSceneGraph based geometry
                        visualisation
  --dataviewer          Even more experimental visualisation of both geometry
                        and data!
  --aimdataviewer       Same as --dataviewer, but will show only the first
                        segment of primary tracks (useful for debugging where
                        your generated particles intersect your geometry).
  -n N, --nevts=N       Simulate N events
  -j N, --jobs=N        Launch N processes [default 1]
  -t, --test            Test geometry consistency and exit
  -l PL, --physlist=PL  Physics List [default QGSP_BIC_HP]
  --showphysicslists    Show available physics lists
  --allowfpe            Do not trap floating point errors
  -s S, --seed=S        Use S as seed for generation of random numbers
                        [default 123456789]
  -v, --visualise       drop to G4 interactive prompt and launch viewer
  -e ENG, --engine=ENG  Use visualisation engine ENG [default OGLSXm]
  -i, --interactive     drop to G4 interactive prompt
  --verbose             enables verbose tracking printouts
  -o FN, --output=FN    Filename for GRIFF output [default tricorder]
  -m MODE, --mode=MODE  GRIFF storage mode [default REDUCED]
```

Fundamentals of geometry description in Geant4





*Info about placement:
- Which is the mother logical volume
- A transform (where is is it located in the mother)
- A copy number (to be able to tell different PV's apart)

Optimised for reuse:
Shapes and materials can go in several logvols
A logvol can be placed again and again

But can get a bit tedious to be so efficient with memory usage when simulating something simple

We provide optional 1-liner: “put this shape here and fill it with this material”:

```
place(new G4Orb("Sample",sample_radius), mat_sample,0,0,sample_posz,lvWorld);
```

We also provide an ever expanding material-database, optimised for neutron facilities

And ways for you to modify parameters of geo and materials without recompilation

geometry_module.cc for TriCorder

GeoTricorder::Construct
Query parameters, compose geometry, return world physical volume

GeoTricorder constructor
Declare configurable parameters with default values

```
GeoTriCorder::GeoTriCorder()
  : GeoConstructBase("G4GeoTriCorder/GeoTriCorder")
{
  addParameterDouble("sample_posz_mm",5.0);
  addParameterDouble("sample_radius_mm",5.0);
  addParameterDouble("detector_size_cm",50.0);
  addParameterDouble("detector_sample_dist_cm",10.0);
  addParameterString("material_sample","ESS_Al");
  addParameterString("material_lab",
                     "IdealGas:formula=0.7*Ar+0.3*CO2{bymass}:ter
}
```

```
G4VPhysicalVolume* GeoTriCorder::Construct()
{
  //Parameters (converting to G4 units immediately as is best practice):
  const double sample_posz = getParameterDouble("sample_posz_mm")*Units::mm;
  const double sample_radius = getParameterDouble("sample_radius_mm")*Units::mm;
  const double det_size = getParameterDouble("detector_size_cm")*Units::cm;
  const double det_depth = 1.0*Units::cm;//ok to hardcode non-interesting parameters
  const double det_sample_dist = getParameterDouble("detector_sample_dist_cm")*Units::cm;
  auto mat_sample = getParameterMaterial("material_sample");
  auto mat_lab = getParameterMaterial("material_la
  auto mat_det = getMaterial("Vacuum");

  //World volume:
  const double dz_world = 1.001 * (std::abs<dou                    ple_radius+det_depth+det_sample
  const double dxy_world = 1.001 * std::max<double>(                radius);
  auto worldvols = place(new G4Box("World", dxy_world,
                         mat_lab,0,0,0,0,INVISIBLE);
  auto lvWorld = worldvols.logvol;

  //Sample:
  place(new G4Orb("Sample",sample_radius),
        mat_sample,0,0,sample_posz,lvWorld,G4Colour(1.0, 1.0, 0.0));

  //Detector:
  place(new G4Box("Detector",0.5*det_size,0.5*det_size,0.5*det_depth),
        mat_det,0,0,sample_posz+det_sample_dist+0.5*det_depth,lvWorld,G4Colour(1.0, 0.0, 0.0));

  return worldvols.physvol;
}
```

Can use standard Geant geometry code here, but preferably take advantage of our material DB, place(..) method and configurable parameters.

```
(thki@tklenovo2014 TriCorder)> ess_tricorder_sim -g
GeoConstructor[G4GeoTriCorder/GeoTriCorder]:
  [dbl] sample_posz_mm = 5
  [dbl] sample_radius_mm = 5
  [dbl] detector_size_cm = 50
  [dbl] detector_sample_dist_cm = 10
  [str] material_sample = "ESS_Al"
  [str] material_lab = "IdealGas:formula=0.7*Ar+0.3*CO2{bymass}:temp_kelvin=293.15:pressure_atm=2.0"
(thki@tklenovo2014 TriCorder)> ess_tricorder_sim material_lab=Vacuum -g
GeoConstructor[G4GeoTriCorder/GeoTriCorder]:
  [dbl] sample_posz_mm = 5
  [dbl] sample_radius_mm = 5
  [dbl] detector_size_cm = 50
  [dbl] detector_sample_dist_cm = 10
  [str] material_sample = "ESS_Al"
  [str] material_lab = "Vacuum"
```

Change and query parameters runtime
From command-line or in python script

Materials

https://ess-ics.atlassian.net/wiki/display/DG/NamedMaterials

Direct definition of materials in Geant4 is Somewhat tedious and error-prone:

```
G4Element* elC  = …;   // define "carbon" element
G4Material* SiO2 = …;  // define "quartz" material
G4Material* H2O = …;   // define "water" material

density = 0.200*g/cm3;
G4Material* Aerog =
   new G4Material("Aerogel",density,ncomponents=3);
Aerog->AddMaterial(SiO2,fractionmass=62.5*perCent);
Aerog->AddMaterial(H2O ,fractionmass=37.4*perCent);
Aerog->AddElement (elC ,fractionmass= 0.1*perCent);
```

Geant4 does provide a database of several common materials based on NIST data. However, it far from completely covers our needs.

To avoid duplication of work (and bugs!), we want to avoid putting such material code directly inside geometry modules!

And we want materials to be free parameters, just as much as other details of a geometry!

Thus, all materials we use can be uniquely defined by a single string, which is parsed and acted upon by our material DB:

- Allowing minimal yet highly functional code in geo modules
- Allowing for easy investigations and scans over material properties
- And allowing for cmd line utilities for materials

```
GeoTriCorder::GeoTriCorder()
  : GeoConstructBase("G4GeoTriCorder/GeoTriCorder")
{
  //...
  addParameterString("material_sample","ESS_Al:temp_kelvin=320");
  //...
}

G4VPhysicalVolume* GeoTriCorder::Construct()
{
  //...
  G4Material * mat_sample = getParameterMaterial("material_sample");
  //...
```

Example of material string definitions

https://ess-ics.atlassian.net/wiki/display/DG/NamedMaterials

- **“G4_xxx” :** Look up xxx in G4's NIST mat DB
- **“NXSG4:nistmat=G4_Al:nxsfile=Al.nxs” :** polycrystalline aluminium, with crystal unit cell info loaded from Al.nxs (**“ESS_Al”** and **“ESS_Cu”** are shortcuts since we use them a lot)
- **"ESS_B4C:b10_enrichment=0.98"** : Enriched boron carbide (98% ^{10}B)
- **"MIX:comp1=CONCRETE:f1=0.99:comp2=Cu:f2=0.01"** : quick and dirty mixture
- **“ESS_POLYETHYLENE”** : custom PE with magic names for TS physics
- **"SHIELDING_paraffin_wax"** : custom paraffin for shielding studies
- **"IdealGas:formula=0.7*Ar+0.3*CO2"** : 70/30 Ar/CO2 gas mixture
- **"IdealGas:formula=He{3:1.0}:pressure_bar=3"** : He3 gas at 3 bar

All materials support parameters for temperature and density. Other parameters depend on the type of material.

- **"IdealGas:formula=0.9*B{10:0.98}F3+0.1*CO2{bymass}:pressure_bar=2:temp_kelvin=300"** :

  90/10 BF3/CO2 by-mass mixture with ^{10}B level enriched to 98% , at 2 bar and 300K

Use the ess_g4materials_namedmat command to see the resulting G4 material dumped, and ess_g4xsectdump_query to extract x-sections and mean-free-path info in the material

Don't start adding material definition code to your geometry module if the above does not fulfill your needs. Rather, get in contact and we will extend the database!

Visualisation

https://ess-ics.atlassian.net/wiki/display/DG/Visualisation+of+Geant4+geometry+and+data

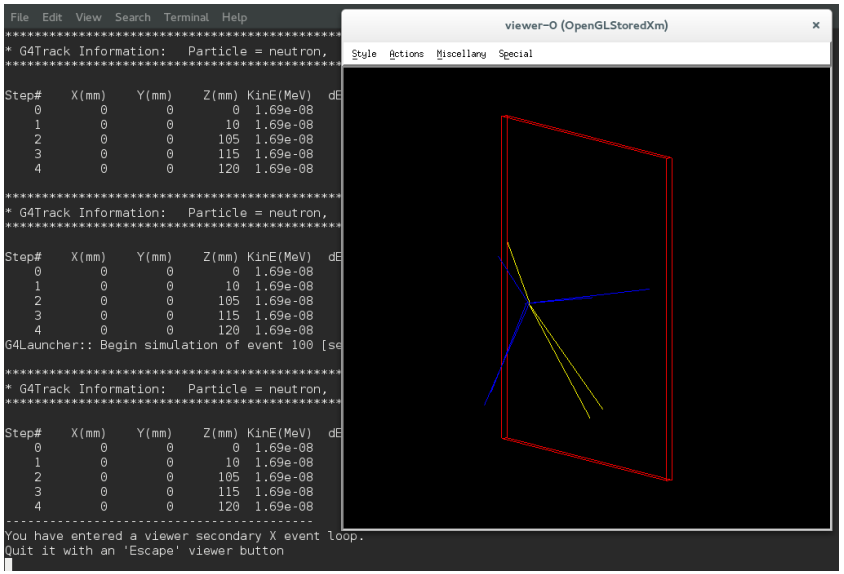
Visualisation of geometry and tracks
is invaluable as a development aid,
debugging tool, and quick view of results.

Bonus points for a visualisation tool if:

- It can generate pretty pictures
- One can easily obtain the desired views
- One can interact with the displayed objects to extract information or change the display

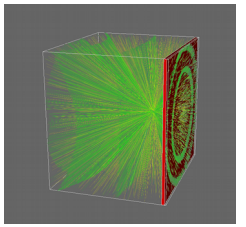
Some people might like the default Geant4 visualisation and interactive command line...

If so, supply “-v” to your simulation script:



Our framework also provides
another option...: “**CoolNameHere**”

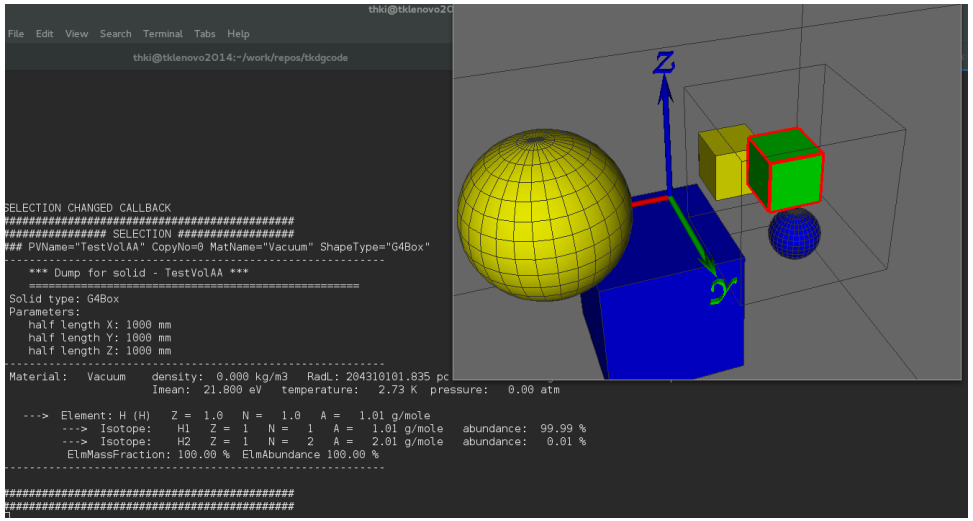
To use it, supply “--viewer” to your sim script.

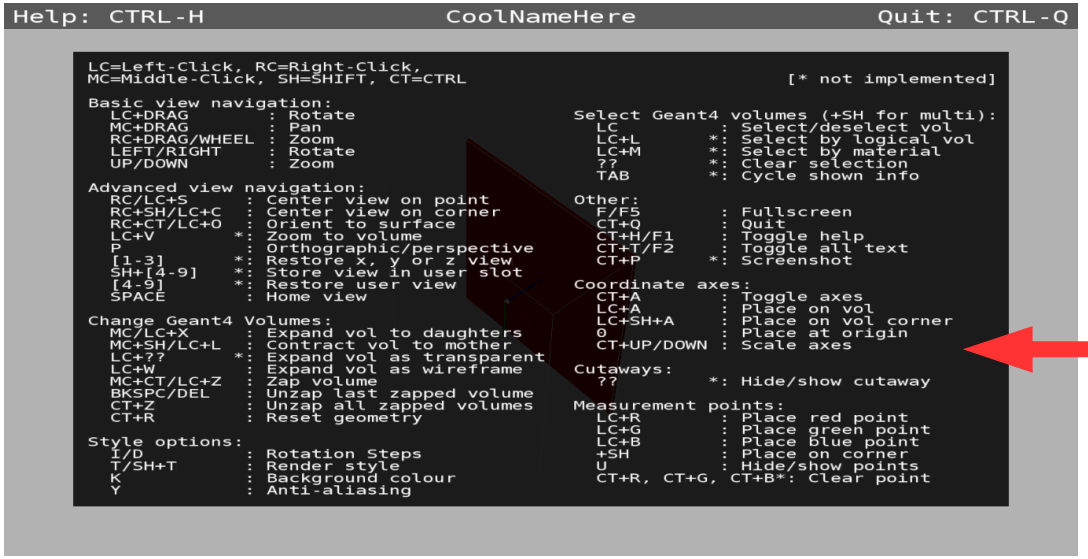


(next slides)

Our viewer (“CoolNameHere”)

A bit rough around the edges in places, but has lots of functionality for geometry debugging:
- interactive open/close vols to get to daughters
- or simply “zapping” of volumes
- Dumps vol/mat info when clicked
- Axes, measurement points, custom data sets



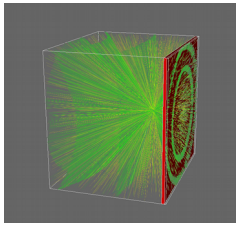


No GUI buttons, but can interact with mouse
And short cut keys (CTRL-H or F1 to see)

Requires OpenSceneGraph (OSG)
But no X11 or gui toolkits.

Easy navigation and display:
- Pan, rotate, zoom with mouse
- Change center to selected point on volume
- Shortcuts to change render style & bgd color
- Shortcut to toggle orthographic/perspective
- Anti-aliasing when hardware supports it

Displays simulated tracks as well
(see later slide)



Generators

Every simulation needs a source of “primary” particles

Picking a generator module

https://ess-ics.atlassian.net/wiki/display/DG/Particle+generators+for+Geant4


```
import G4StdGenerators.SimpleGen as genmodule
gen = genmodule.create()
gen.particleName = 'neutron'
gen.neutron_wavelength_aangstrom = 2.2
```

For most projects, a pre-existing generator module can simply be re-used (at least to start)

```
(thki@tklenovo2014 TriCorder)> ess_tricorder_sim -p
ParticleGenerator[G4StdGenerators/SimpleGen]:
  [dbl] fixed_x_meters = 0
  [dbl] fixed_y_meters = 0
  [dbl] fixed_z_meters = 0
  [dbl] fixed_energy_eV = (
  [dbl] neutron_wavelength_
  [dbl] fixed_xdir = 0
  [dbl] fixed_ydir = 0
  [dbl] fixed_zdir = 1
  [str] particleName = "neutron"
```

As for geometry modules, parameters can be queried and modified at runtime

```
(thki@tklenovo2014 TriCorder)> ess_tricorder_sim fixed_zdir=-1 -p
ParticleGenerator[G4StdGenerators/SimpleGen]:
  [dbl] fixed_x_meters = 0
  [dbl] fixed_y_meters = 0
  [dbl] fixed_z_meters = 0
  [dbl] fixed_energy_eV = 0.016901694474302
  [dbl] neutron_wavelength_aangstrom = 2.2
  [dbl] fixed_xdir = 0
  [dbl] fixed_ydir = 0
  [dbl] fixed_zdir = -1
  [str] particleName = "neutron"
```

Utility for playing around with generator modules directly (i.e. without a sim script):

```
ess_g4utils_querygenerator -g G4StdGenerators.ProfiledBeamGen -v -n2000 \
        spread_x_mm=500 spread_y_mm=300 spread_mode=FLATCIRCULAR
```

Generating from external sources

with a simple interface which can read and shoot particles found in an MCNP file
format to ascii format via the command ess_mcnputils_mcnp2ascii):

Note that the input ASC… it has options to rotate and
translate, in case the MCNP a…

```
$> ess_g4utils_querygenerator -g…
ParticleGenerator[G4MCNPGun/MCNPGun]…
  [str] input_file = "mcnpdata.txt.gz"
  [dbl] dx_meter = 0
  [dbl] dy_meter = 0
  [dbl] dz_meter = 0
  [dbl] rotx_degree = 0
  [dbl] roty_degree = 0
  [dbl] rotz_degree = 0
```

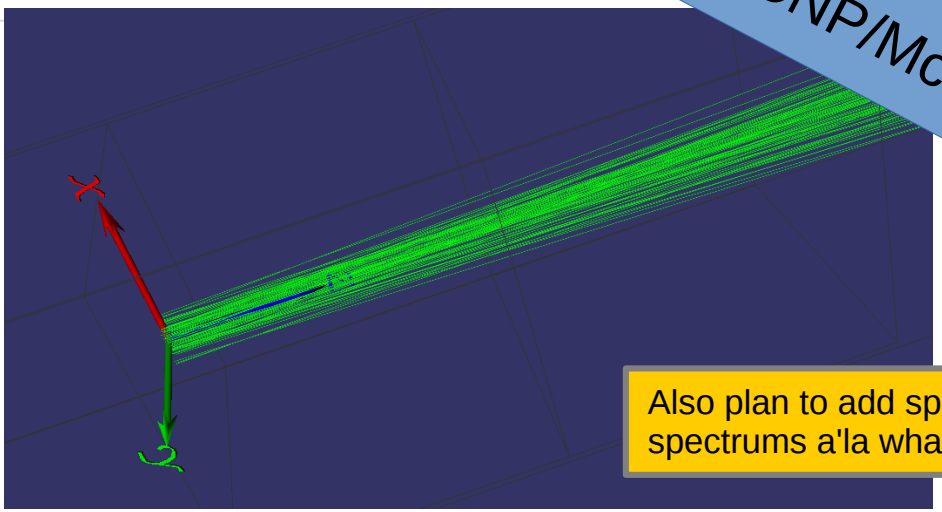
So far, most target station simulations are done in MCNP and most instrument simulations in McStas (excl. shielding studies of course).
So quite useful to be able to catch their output and continue downstream simulations with Geant4.

So far we have generators for MCNP and McStas, …t could easily add more if required:

…tributions in histograms?

OBSOLETE METHOD

(better solution now exists for exchanging data with MCNP/McStas – see MCPL talk)



Also plan to add specialised sources (like ESS spectrums a'la what is in McStas) as we go along.

Want full flexibility but no hassle? A generator in python is for you! :-)

Recommended!

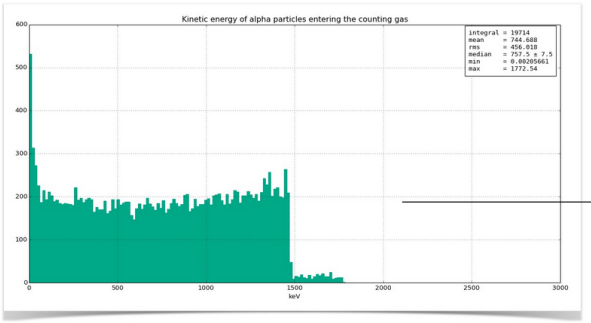
```
class MySillyGen(G4CustomPyGen.GenBase):
    def generate_event(self,gun):
        gun.set_type('gamma')
        gun.set_direction(0,0,1)
        gun.set_position(self.randGauss()*2.0*Units.cm,0,0)
        gun.set_energy(self.rand(0.5,2.0)*Units.MeV)

gen = MySillyGen()
```

Just a handful of lines to add **directly in your simulation script** for a quick test

Perhaps add a few tunable parameters and a bit of initialisation

```
class GammaExpGen(G4CustomPyGen.GenBase):
    """Generator which produces a beam of gammas whose energy is given by an
    exponential distribution with a user supplied average"""

    def declare_parameters(self):
        self.addParameterDouble("average_energy_keV",50.0)

    def init_generator(self,gun):
        gun.set_type('gamma')
        gun.set_direction(0,0,1)
        gun.set_position(0,0,0)

    def generate_event(self,gun):
        gun.set_energy(self.randExponential() * self.average_energy_keV*Units.keV)
```

Might generate based on distributions in histograms...



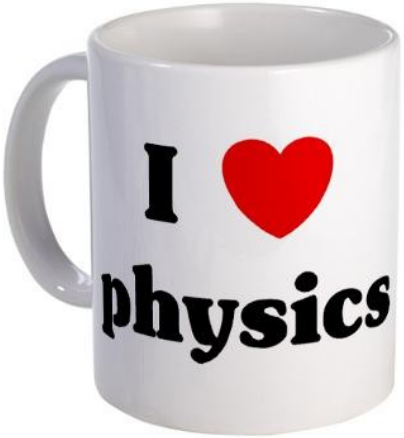

```
class AlphaHistGen(G4CustomPyGen.GenBase):
    """Generator which produces an alpha particle with an energy sampled from a histogram"""

    def declare_parameters(self):
        self.addParameterString('energy_histogram',
                                'G4CustomPyGen/example.shist:alpha_energy:keV')

    def init_generator(self,gun):
        gun.set_type('alpha')
        gun.set_direction(0,0,1)
        gun.set_position(0,0,0)
        self._esampler = self.create_hist_sampler(self.energy_histogram)

    def generate_event(self,gun):
        gun.set_energy(self._esampler())
```

Physics

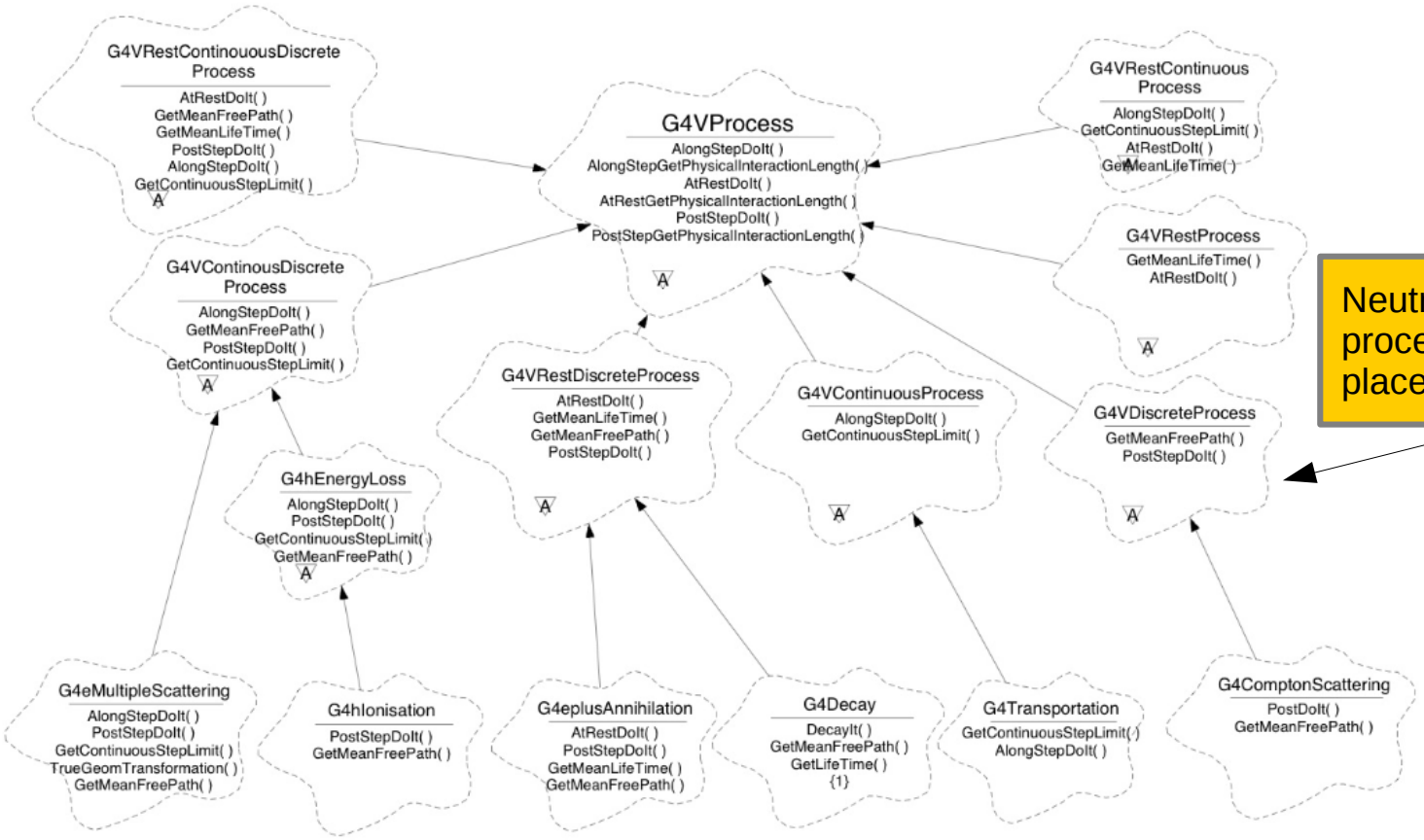
Selecting G4 physics lists, using our neutron-extensions

Physics!!

REFERENCE



"Laws of physics" are implemented in Geant4 in terms of processes, and each particle type has an associated list of processes:



Neutrons mainly have "discrete" processes, which either take place completely or not at all

In reality, a process is just a C++ class, so anyone can in principle add their own physics modelling to Geant4 (and we are doing so, see following slides!)

Physics lists

A consistent collection of particle types and their associated processes Is a **physics list**. Creating (and validating!) one of these can be a bit tricky, but luckily Geant4 provides a long list of reference lists. On top of those, we provide some ourselves.

To see available physics lists, supply "--showphysics" to any sim script, or run ess_g4physicslists_showall :

Note that I hid 102 lists in the output, so it would fit on the slide.
Since all G4 lists come in flavours depending on EM physics (e.g. GQSP_BIC_HP_*LIV*)

```
(thki@tklenovo2014 tkdgcode)> ess_g4physicslists_showall |egrep -v '_EM|_PEN|_LIV'
Available physics lists:
   FTFP_BERT [Geant4 reference list]
   FTFP_BERT_HP [Geant4 reference list]
   FTFP_BERT_TRV [Geant4 reference list]
   FTFP_INCLXX [Geant4 reference list]
   FTFP_INCLXX_HP [Geant4 reference list]
   FTF_BIC [Geant4 reference list]
   LBE [Geant4 reference list]
   QBBC [Geant4 reference list]
   QGSP_BERT [Geant4 reference list]
   QGSP_BERT_HP [Geant4 reference list]
   QGSP_BIC [Geant4 reference list]
   QGSP_BIC_HP [Geant4 reference list]
   QGSP_FTFP_BERT [Geant4 reference list]
   QGSP_INCLXX [Geant4 reference list]
   QGSP_INCLXX_HP [Geant4 reference list]
   QGS_BIC [Geant4 reference list]
   Shielding [Geant4 reference list]
   ESS_Empty [List defined in package G4PhysicsLists]
   ESS_FTFP_BERT_HP_TS [List defined in package G4PhysListsTS]
   ESS_FTFP_INCLXX_HP_TS [List defined in package G4PhysListsTS]
   ESS_QGSP_BERT_FHP_TS [List defined in package G4PhysListsFHP_ACE]
   ESS_QGSP_BERT_HP_ACE [List defined in package G4PhysListsFHP_ACE]
   ESS_QGSP_BERT_HP_TS [List defined in package G4PhysListsTS]
   ESS_QGSP_BIC_FHP_TS [List defined in package G4PhysListsFHP_ACE]
   ESS_QGSP_BIC_HP_ACE [List defined in package G4PhysListsFHP_ACE]
   ESS_QGSP_BIC_HP_TS [List defined in package G4PhysListsTS]
   ESS_QGSP_INCLXX_HP_TS [List defined in package G4PhysListsTS]
   ESS_Shielding_TS [List defined in package G4PhysListsTS]
```

Our current default

G4's ref lists

Very important for neutrons < 20MeV to use "_HP" list!

Quick guide to picking a physics model here:

http://geant4.slac.stanford.edu/MIT2015/Physics1MIT.pdf

Our lists

- **TS**: Enable G4's own thermal scattering implementation
- **ACE** : XX's TS impl. Loading MCNP ACE files.
- **FHP** : XX's fast and precise "_HP" (faster shielding studies!)
- **ESS_Empty** : Nada!

+ NXSG4 for polycrystals
+ project lists (ShieldingCuts, Scint)

Picking a physics list in our framework

There's nothing too it, really!

Firstly, you can change the default In your simulation script:

```
launcher.setPhysicsList(“ESS_QGSP_BIC_HP_TS”)
```

secondly, you can override at the command line:

```
$>  ess_tricorder_sim -lESS_QGSP_BIC_HP_TS
```

Investigating cross-sections (1/2)

https://ess-ics.atlassian.net/wiki/display/DG/Extract+and+investigate+cross-sections+from+Geant4

Very often, one wonders about x-sections and mean-free-paths:
- Either because one need the info for some reason
- Or because one wants to check if some physics of interest is implemented in Geant4 in a reasonable way.

This info is not normally easy to extract in Geant4, but we provide a special hook which can reliably extract the cross-sections at run-time

Option 1: Supply -x to your simulation script and get x-sect info dumped for all combinations of materials and particle types encountered during the course of event simulation:

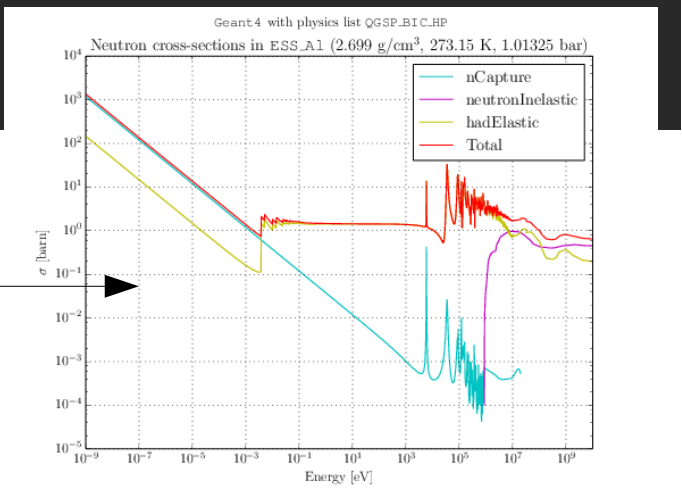
```
G4Launcher:: Begin simulation of event 1 [seed 123]
G4Launcher::  XSectSpy: Writing X-section file xsects_discreteprocs_neutron__ESS_Al__QGSP_BIC_HP.txt
G4Launcher::  XSectSpy: Writing X-section file xsects_discreteprocs_neutron__IdealGas:formula=0.7*Ar+0.3*CO2{bymass}:temp_kelvin=293.15:p
G4Launcher::  XSectSpy: Writing X-section file xsects_discreteprocs_Al27__ESS_Al__QGSP_BIC_HP.txt
G4Launcher:: Begin simulation of event 2 [seed 473794492965156728]
G4Launcher::  XSectSpy: Writing X-section file xsects_discreteprocs_neutron__Vacuum__QGSP_BIC_HP.txt
G4Launcher:: Begin simulation of event 3 [seed 6558893583356662305]
G4Launcher:: Begin simulation of event 4 [seed 7020782182520143055]
G4Launcher:: Begin simulation of event 5 [seed 11338721050658049671]
G4Launcher:: Begin simulation of event 6 [seed 4001671858511099590]
G4Launcher:: Begin simulation of event 7 [seed 10337799263997629515]
G4Launcher:: Begin simulation of event 8 [seed 8101454701794766284]
G4Launcher:: Begin simulation of event 9 [seed 47203770351912269 47]
G4Launcher:: Begin simulation of event 10 [seed 41957998923589004 18]
G4Launcher:: Simulation done
```

ess_tricorder_sim -x

Afterwards you can either plot the files with:

"ess_xsectparse_plotfile <filename>"

or you can load the curves into python if you have some special analysis in mind.

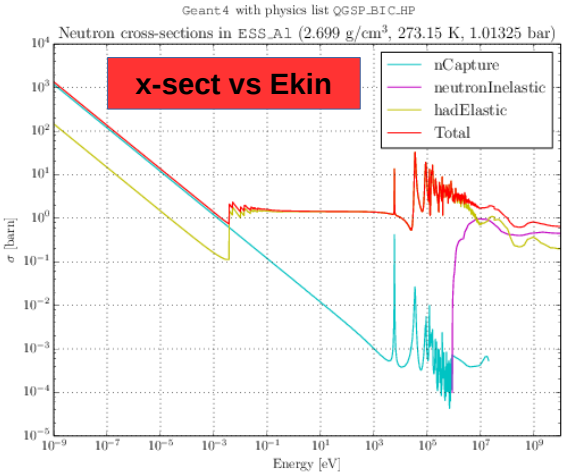


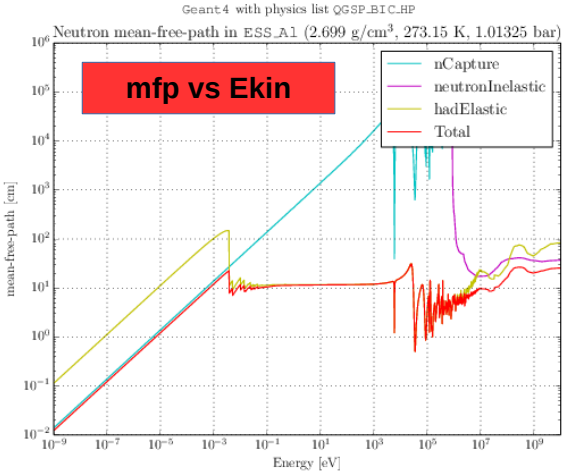
Investigating cross-sections (2/2)

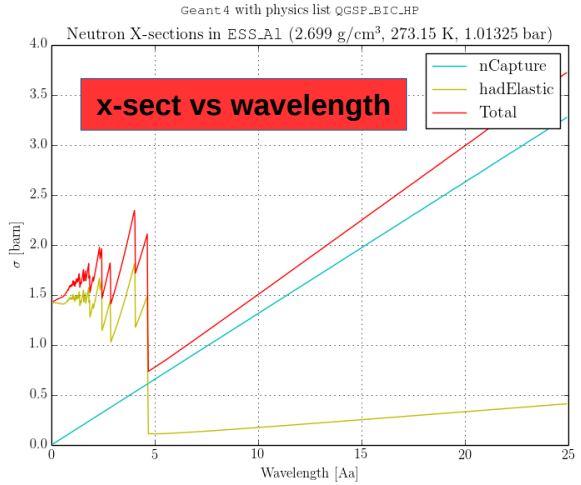
https://ess-ics.atlassian.net/wiki/display/DG/Extract+and+investigate+cross-sections+from+Geant4

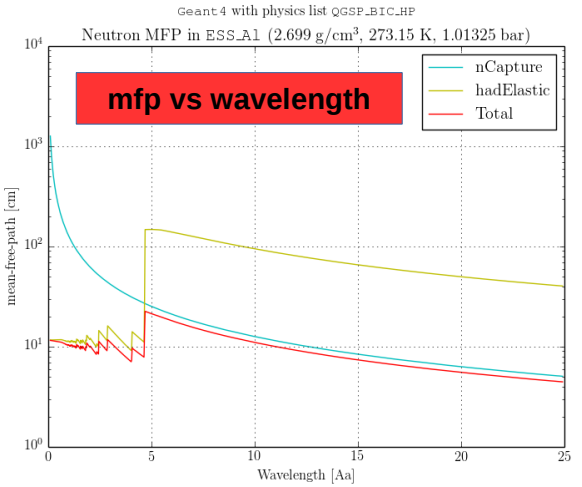
Option 2: Use the ess_g4xsectdump_query script to directly extract info you want, with no need for a sim-script.


```
$>  ess_g4xsectdump_query -pneutron -lQGSP_BIC_HP -mESS_Al [-w]
```





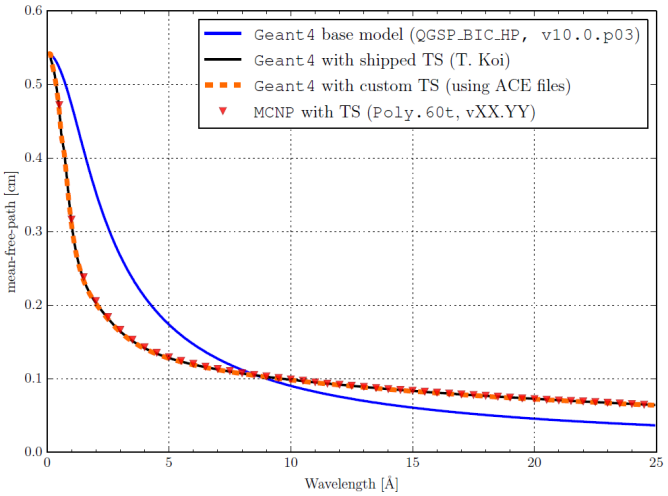




Thermal scattering physics

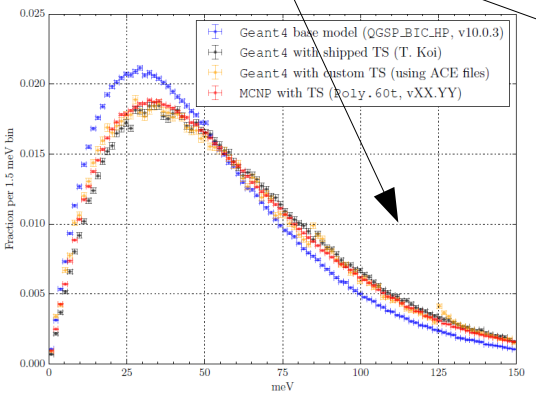
Chemical bindings have very significant effects on interaction cross-sections of neutrons at thermal energies, but Geant4 by default treats all materials with a free gas approximation.

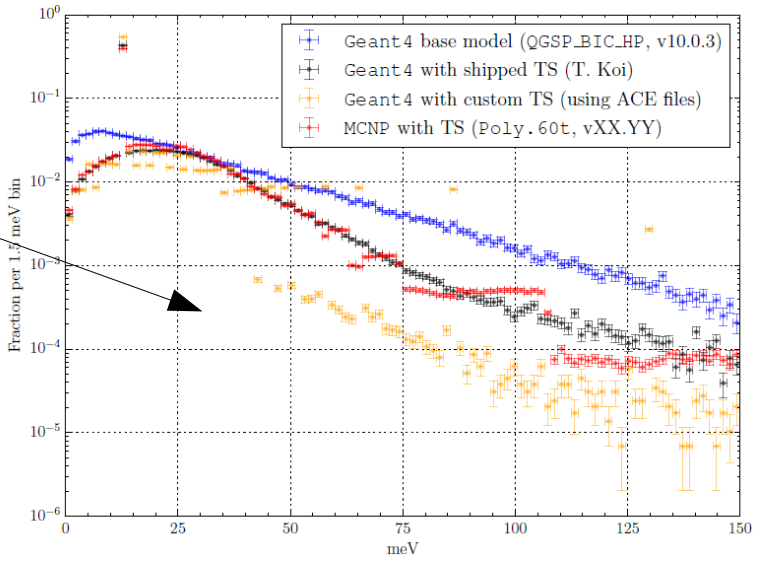
Example: Mean-free-path in polyethylene for 1.8Aa neutrons change by a factor of ~2 when the chemical bindings of the hydrogen atoms are included!



Requires custom physics lists and custom materials in order to work, and only work for ~10 materials
In Geant4's own implementation → value in having common material DB and physics lists.

Comparisons between the _TS, _ACE and MCNP models, reveal discrete artifacts of implementation. Xiao Xiao plan to solve this by developing new TS models (which would also work for many more materials).

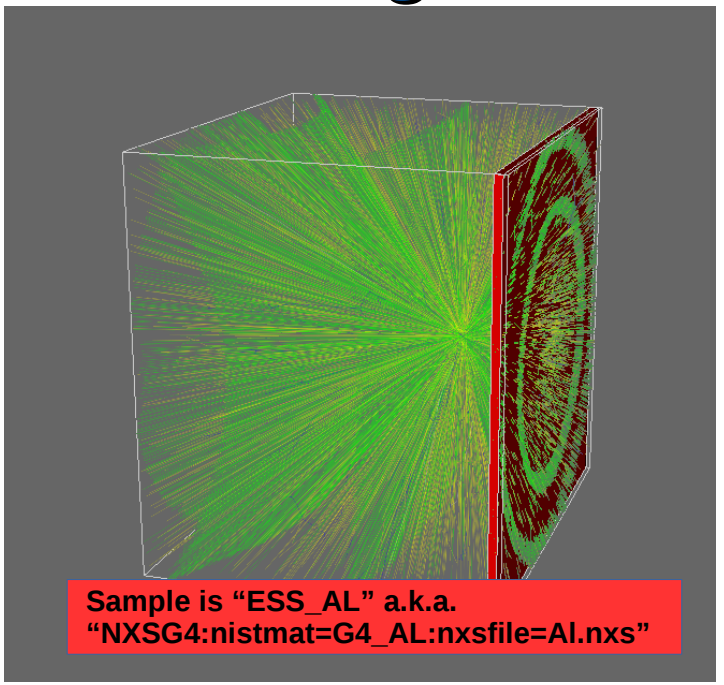


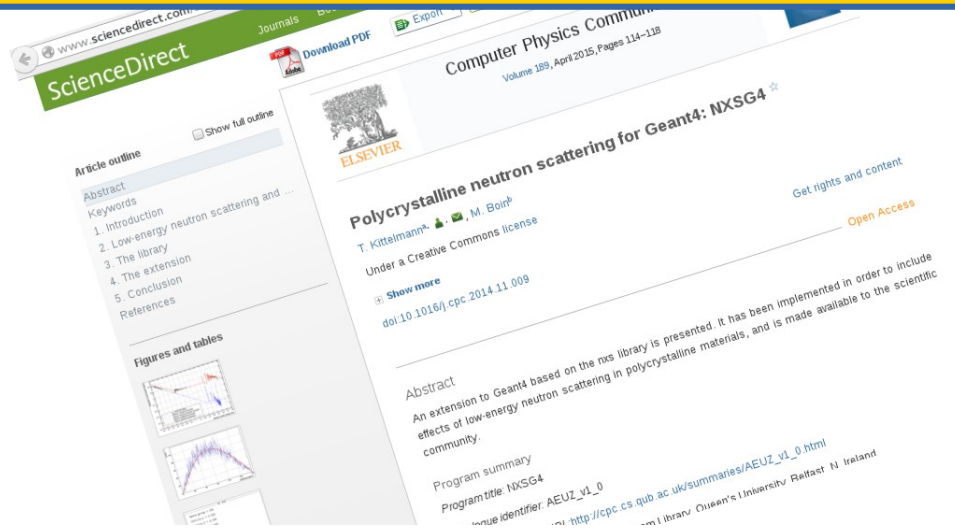


NXSG4: Polycrystalline/powder scattering

Our own, published extension, which enables poly-crystalline scattering in Geant4.

– plugs dynamically into any physics list on demand, so does not influence list choice

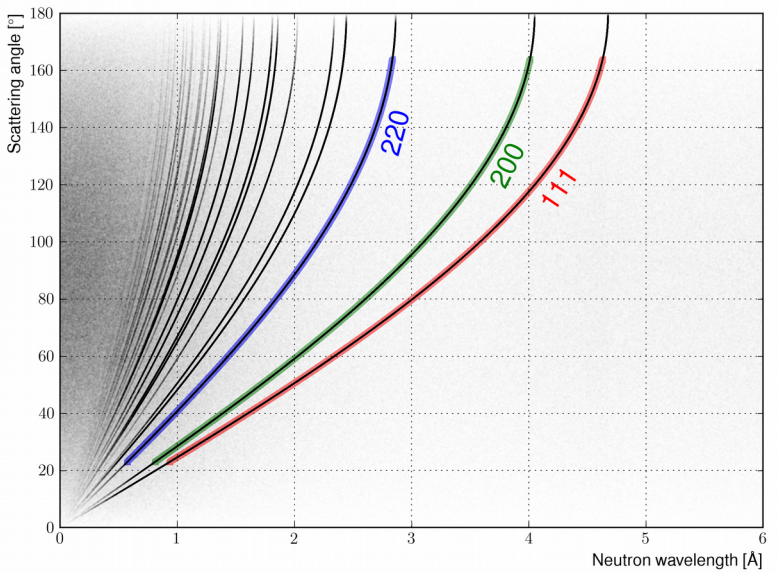




Al.nxs

```
space_group = 225
lattice_a = 4.049
lattice_b = 4.049
lattice_c = 4.049
lattice_alpha = 90
lattice_beta = 90
lattice_gamma = 90
debye_temp = 429.0
add_atom = Al 3.449 0.008 0.23 26.98 0.0 0.0 0.0
```

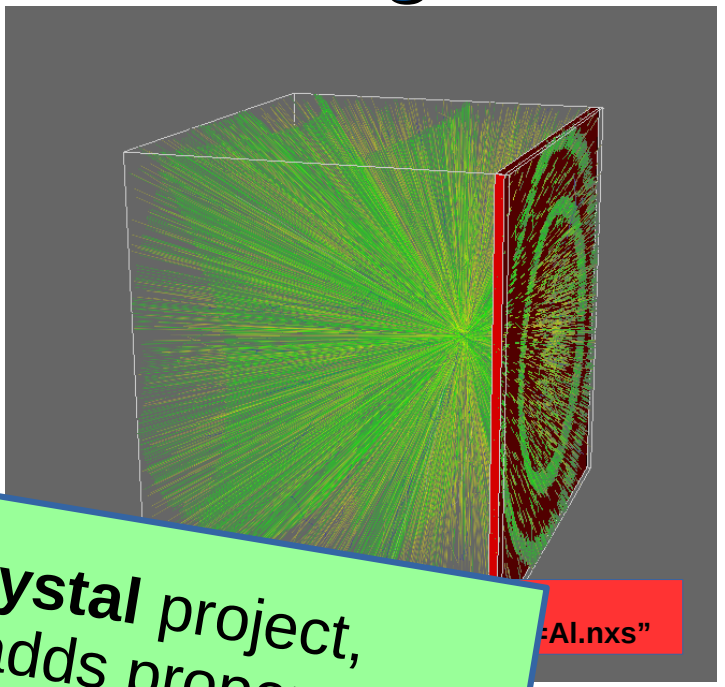
This is how to define the Crystal structure



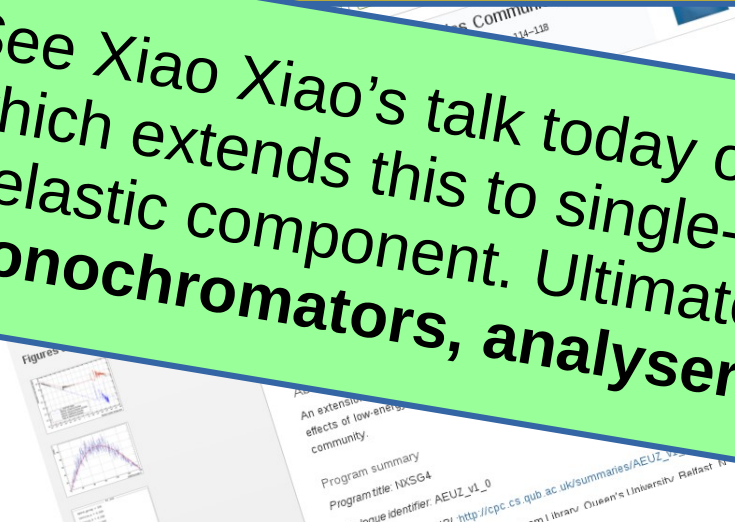
NXSG4: Polycrystalline/powder scattering

Our own, published extension, which enables poly-crystalline scattering in Geant4.

– plugs dynamically into any physics list on demand, so does not influence list choice



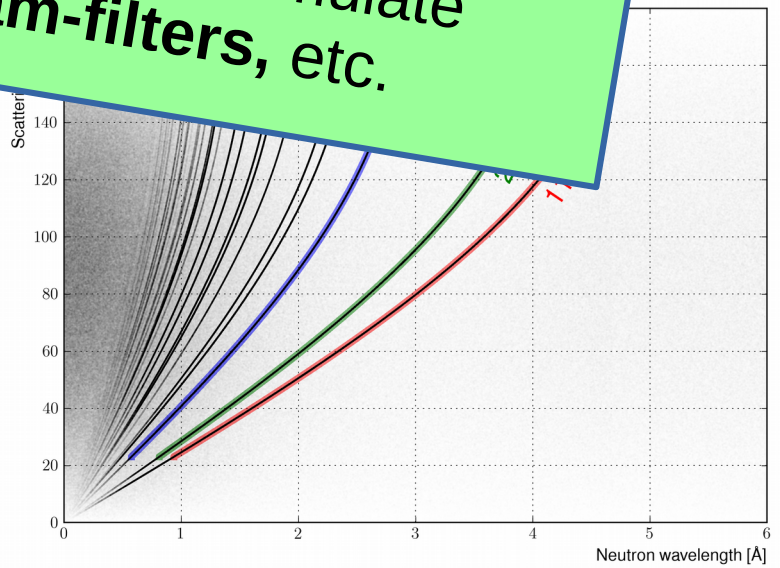
See Xiao Xiao's talk today on the **NCrystal** project, which extends this to single-crystal + adds proper inelastic component. Ultimately, so we can simulate **monochromators, analysers, beam-filters,** etc.



Al.nxs

```
space_group = 225
lattice_a = 4.049
lattice_b = 4.049
lattice_c = 4.049
lattice_alpha = 90
lattice_beta = 90
lattice_gamma = 90
debye_temp = 429.0
add_atom = Al 3.449 0.008 0.23 26.98 0.0 0.0 0.0
```

This is how to define the Crystal structure



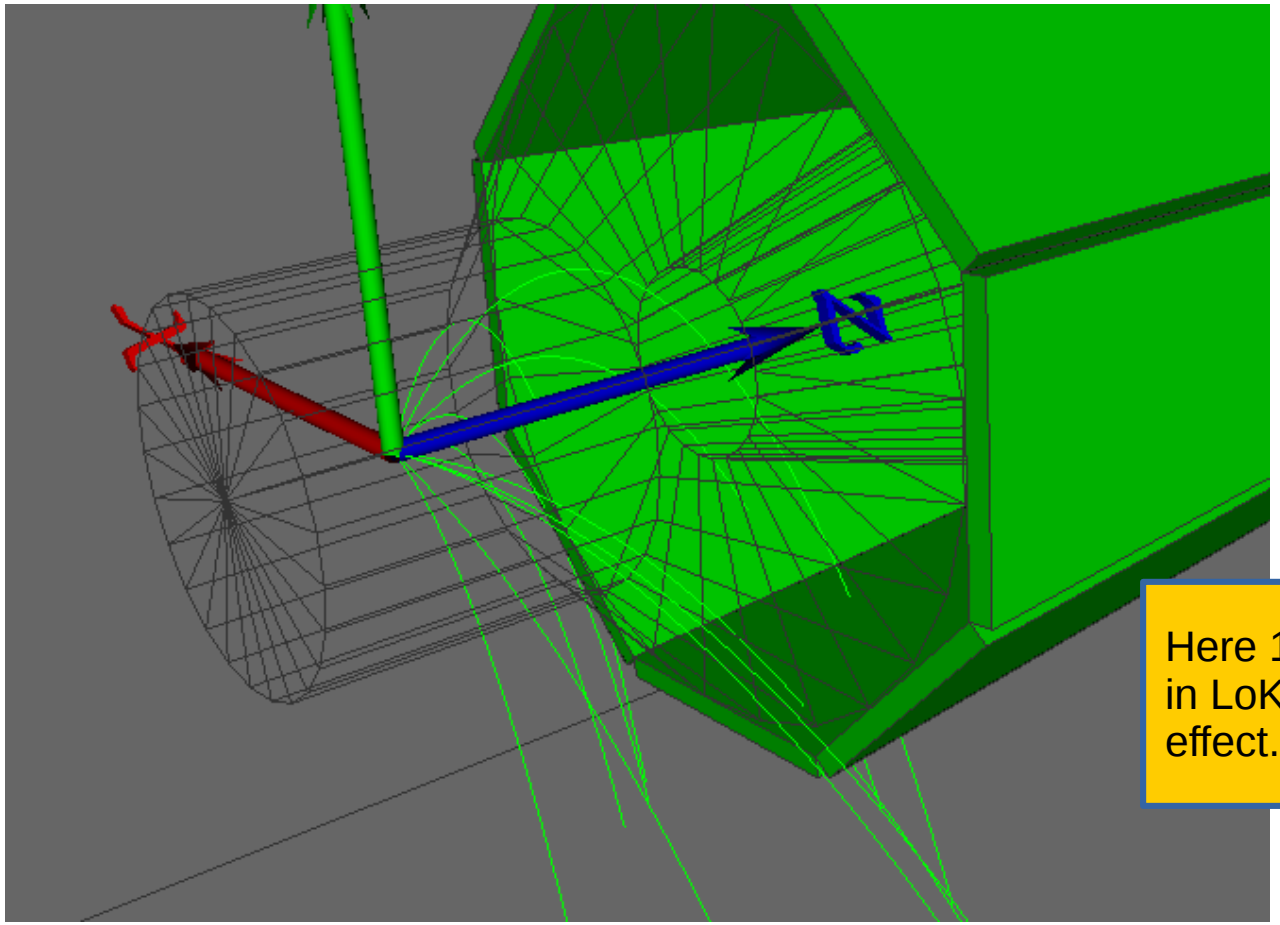
Gravity for cold beam-lines

https://ess-ics.atlassian.net/wiki/display/DG/EnablingGravityInGeant4

Just add two lines to your sim-script:

```
import G4GravityHelper.NeutronGravity as ng
ng.enableNeutronGravity(launcher,-1,0,0,g=9.82)
```

(defaults to Y-axis pointing up if no direction specified)

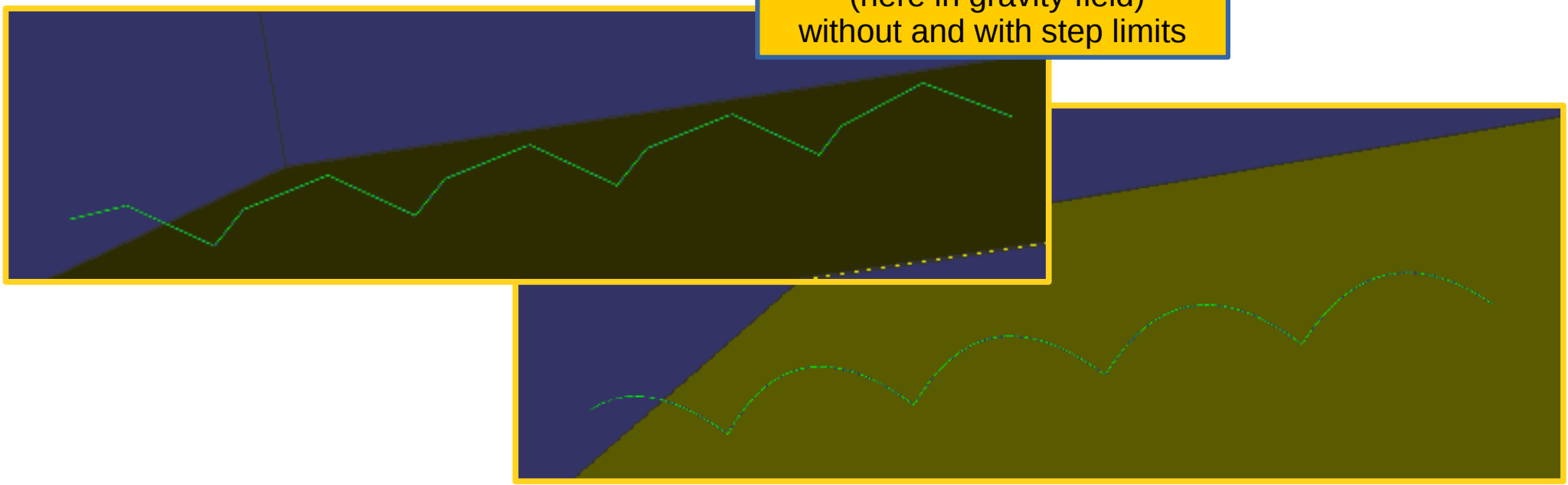


Here 1000Å neutrons in LoKI for clear visual effect...

Other ongoing work: step-limits & mirrors

DGSW-265 & DGSW-288

Super-mirror physics
(here in gravity field)
without and with step limits



Limit step-size for certain particles in certain volumes:

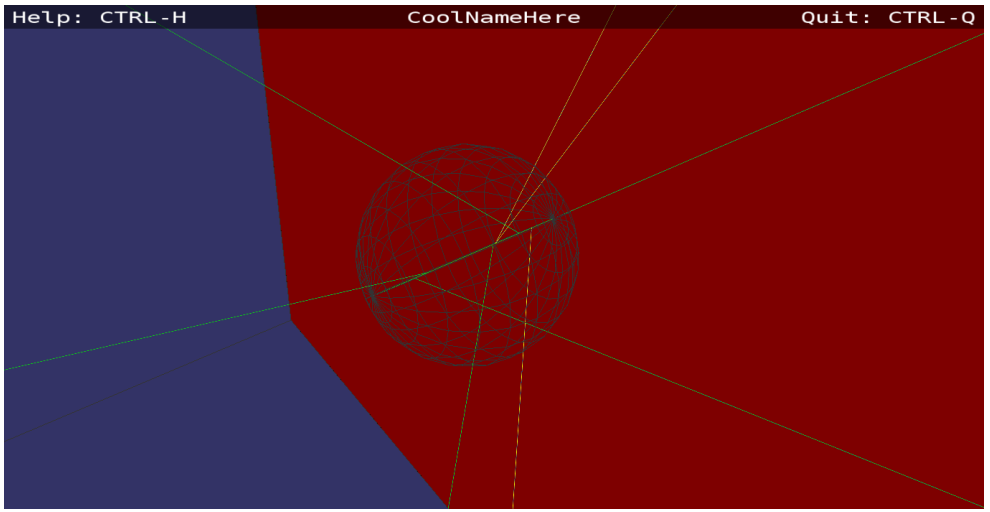
```
##############################################################
# Limit step-size for certain particles in certain volumes
from Core.Units import mm

import G4StepLimitHelper.helper as slh
slhelper = slh.G4StepLimitHelper ()
slhelper.addLimit(11, 'CountingGas', 0.1*mm) #e-
```

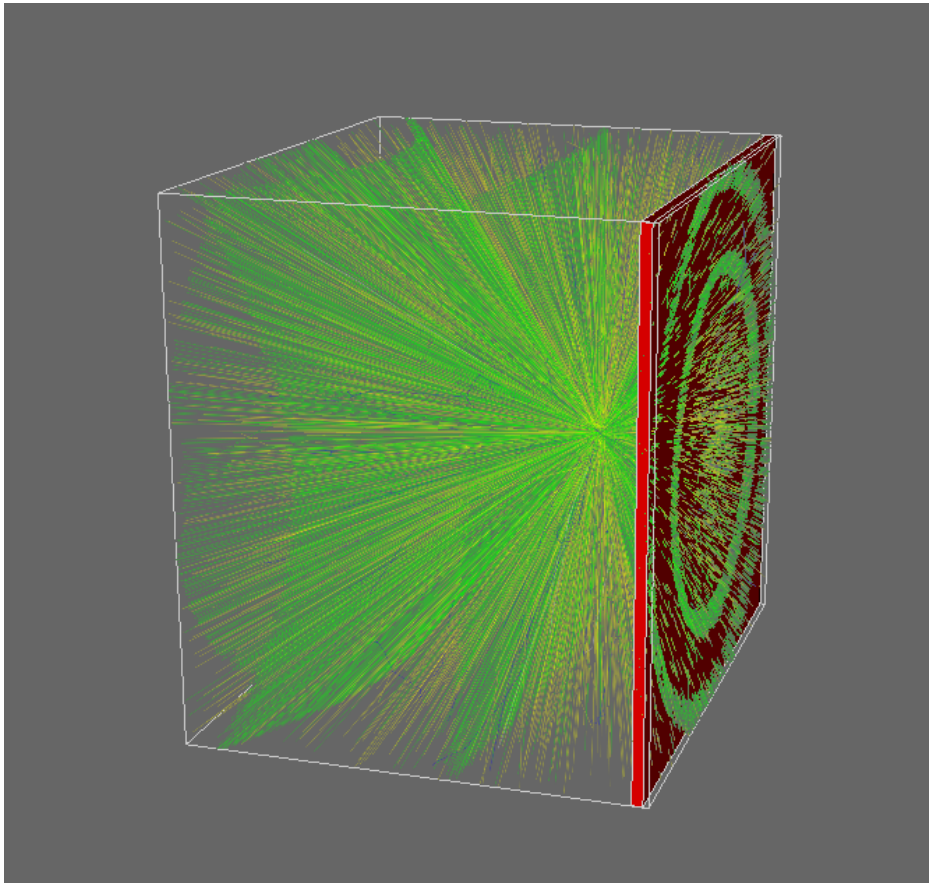
Data extraction & analysis

Event inspection, GRIFF, HeatMaps, custom hooks, pylab/matplotlib

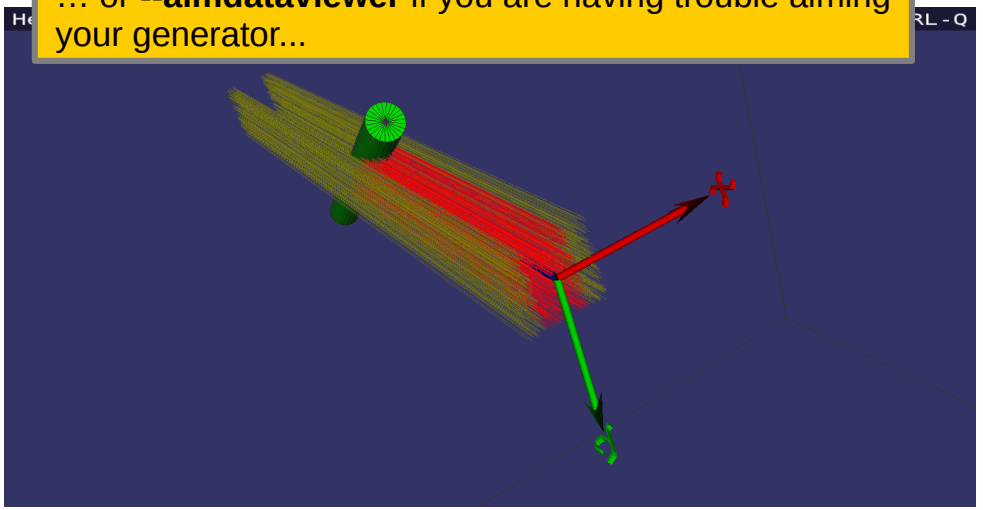
Visualising simulated tracks

Supply **--dataviewer** to the sim script, using **-n<NEVTS>** to control statistics





… or **--aimdataviewer** if you are having trouble aiming your generator...



The data display is a bit basic, and has no interactive features yet – feedback appreciated as always!

Can also view tracks in GRIFF files (even with geometry!):
$> ess_g4osg_viewgriff --loadgeo myresults.griff

Inspecting at command-line

Less glamorous than 3D, but highly useful when short steps are be hard to see in the 3D viewer

To activate, supply **--verbose** or **-r** to sim-script. Specifying multiple times increases verbosity.

```
G4Launcher:: Begin simulation of event 1 [seed 1234567898]

*********************************************************************************************************
* G4Track Information:   Particle = neutron,   Track ID = 1,   Parent ID = 0
*********************************************************************************************************

Step#    X(mm)    Y(mm)    Z(mm) KinE(MeV)  dE(MeV) StepLeng TrackLeng  NextVolume ProcName
    0        0        0        0  1.69e-08        0        0         0      Sample initStep
    1        0        0      3.3  1.67e-08        0      3.3       3.3      Sample hadElastic
    :----- List of 2ndaries - #SpawnInStep=  1(Rest= 0,Along= 0,Post= 1), #SpawnTotal=  1 --------------
    :         0         0       3.3  4.47e-08               Al27
    :------------------------------------------------------------------ EndOf2ndaries Info --------------
    2   0.0346     4.77      6.5  1.67e-08        0     5.74      9.05       World Transportation
    3      1.1      152      105  1.67e-08        0      177       186    Detector Transportation
    4     1.21      166      115  1.67e-08        0     17.9       204       World Transportation
    5     1.26      174      120  1.67e-08        0     9.19       213  OutOfWorld Transportation

*********************************************************************************************************
* G4Track Information:   Particle = Al27,   Track ID = 2,   Parent ID = 1
*********************************************************************************************************

Step#    X(mm)    Y(mm)    Z(mm) KinE(MeV)  dE(MeV) StepLeng TrackLeng  NextVolume ProcName
    0        0        0      3.3  4.47e-08        0        0         0      Sample initStep
    1 -1.45e-07 2.63e-09      3.3         0 4.47e-08 2.17e-07  2.17e-07      Sample ionIoni
G4Launcher:: Begin simulation of event 2 [seed 17253458942787885784]

*********************************************************************************************************
* G4Track Information:   Particle = neutron,   Track ID = 1,   Parent ID = 0
*********************************************************************************************

Step#    X(mm)    Y(mm)    Z(mm) KinE(MeV)  dE(Me
```

Example output from: **`ess_tricorder_sim -rr`**

ProTip: Use with ESS_Empty physics list to debug your generator aim!

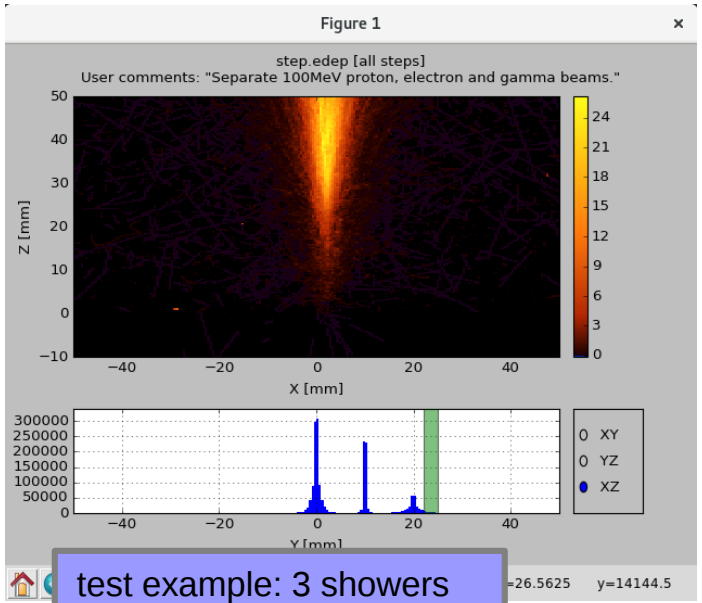
```
$>  ess_tricorder_sim -lESS_Empty --verbose
```

Heat-maps

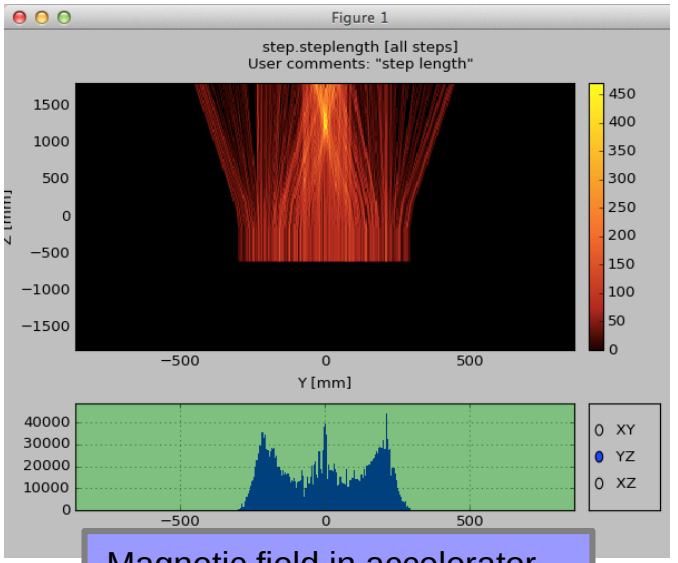
https://ess-ics.atlassian.net/wiki/display/DG/G4HeatMap

- Sum quantities in virtual 3D mesh
- Can handle large number of bins efficiently
- Our new ExpressionParser makes it simple to specify:
  - Quantity to be summed
  - For which steps to do the summing
- Interactive view of results with ess_mesh3d_browse
- **All without touching a single line of code!**

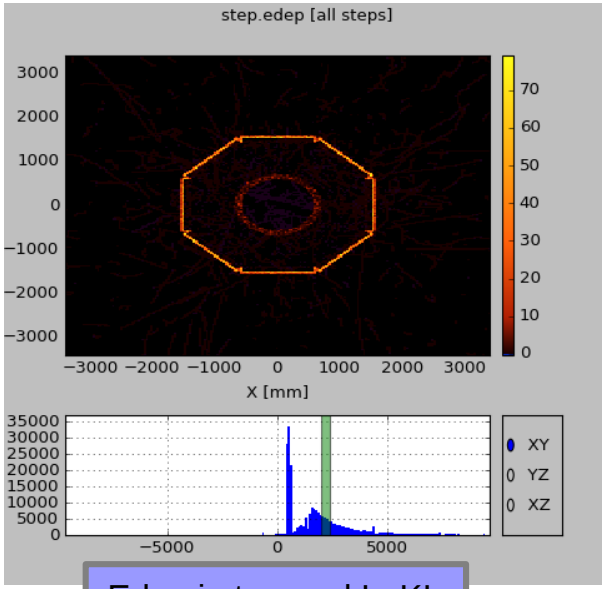
```
ess_myproj_sim --heatmap [edep of all steps]
ess_myproj_sim --heatmap=”step.steplength*step.ekin” [energy flux]
ess_myproj_sim --heatmap=”step.edep where trk.is_photon” [edep of photons]
ess_myproj_sim --heatmap=help [print detailed instructions]
```



test example: 3 showers



Magnetic field in accelerator



Edep in tapered LoKI

Code-based access to results

Normally analysis in Geant4 will be done by registering call-back functions, which gets called for each step particles take through geometry:

```
void myAnalysisFunction(G4Step*step) {
   //Do stuff here with the step, fill
   //variables, histograms, whatever...
};
```

In practice this might be a *Sensitive Detector* or perhaps a *Stepping Action*

This might be adequate for some purposes, and certainly it can be the most CPU efficient manner af accessing the results.

In our framework, we do this by making so-called "custom hooks". However, it is rarely needed.

However, there are some potential drawbacks:

- Need to rerun expensive Geant4 simulation if analysis changes or bugs are found.
- Order steps are delivered might change as Geant4 evolve with multi-threading and vectorisation (SIMD) support, so some analyses might break.
- We are getting fed the results step-by-step, lacking the global overview
  - like observing a painting 1cm^2 at a time *(hey, at least it wasn't a car analogy!)*
  - so some questions might be difficult (or error-prone) to answer (what is the origin of these weird electrons over here?!?)
- Any kind of data (incl. job metadata) you want stored requires you have to go through the hassle of extracting and storing it yourself.

All, in all, perhaps not so efficient in terms of manpower as it can be in terms of CPU...

So therefore we provide...

GRIFF – Geant4 Results in Friendly Format

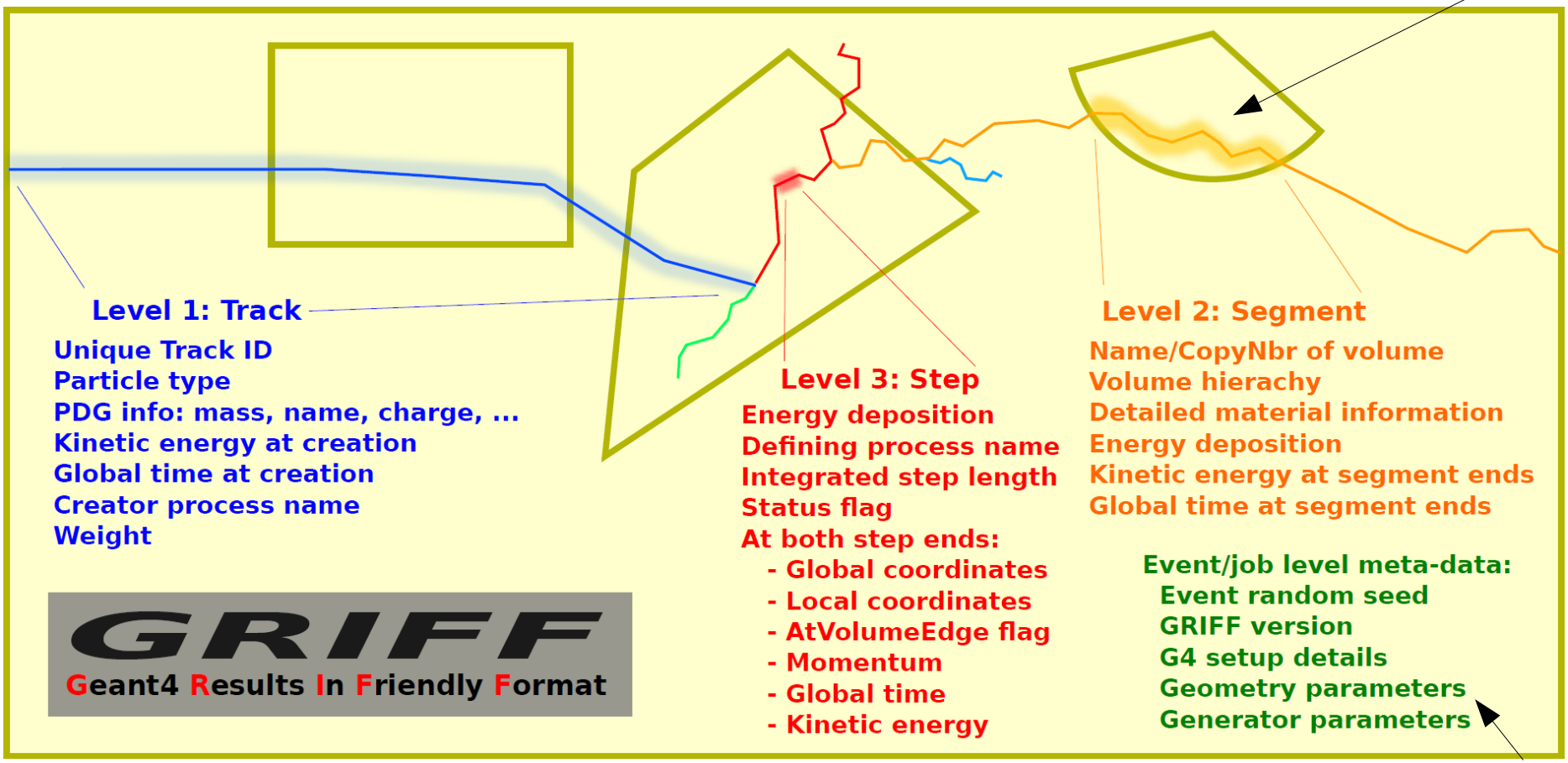
https://ess-ics.atlassian.net/wiki/display/DG/Griff

REFERENCE

A custom output format containing entire events

Allows object-oriented analysis code, with convenient links between objects
(mother<->daughters, trk<->segments<->steps)

Introduces convenient "segment" concept



Level 1: Track

- Unique Track ID
- Particle type
- PDG info: mass, name, charge, ...
- Kinetic energy at creation
- Global time at creation
- Creator process name
- Weight

Level 2: Segment

- Name/CopyNbr of volume
- Volume hierachy
- Detailed material information
- Energy deposition
- Kinetic energy at segment ends
- Global time at segment ends

Level 3: Step

- Energy deposition
- Defining process name
- Integrated step length
- Status flag
- At both step ends:
  - Global coordinates
  - Local coordinates
  - AtVolumeEdge flag
  - Momentum
  - Global time
  - Kinetic energy

Event/job level meta-data:

- Event random seed
- GRIFF version
- G4 setup details
- Geometry parameters
- Generator parameters

GRIFF
Geant4 Results In Friendly Format

No need for custom book-keeping of meta-data!

Three output modes (*:default):
FULL : Write all steps
REDUCED*: 1 merged step per segment
MINIMAL : No steps, just tracks and segments
Option to add custom output filters.

Reading does not depend on Geant4, shielding your analysis from Geant4 development changes.

Highly optimised For speed and storage size

Enable GRIFF output in our framework

It is actually on by default... (set output to “none” to disable).

As always, you can change the default In your simulation script:

```
launcher.setOutput(“tricorder”,”FULL”)
```

and you can override at the command line:

```
$>  ess_tricorder_sim -otricorder.griff -mFULL
```

Example GRIFF analysis

Tricorder/app_ana/main.cc becomes the command *ess_tricorder_ana*

If you wish, Griff will deal with the command line arguments, loading the requested files

```
#include "GriffAnaUtils/All.hh"

int main(int argc,char**argv) {

  GriffDataReader dr(argc,argv);

  printf("Loading file with boron thickness of %g micron\n",
         dr.setup()->geo().getParameterDouble("boronThickness_micron"));

  while (dr.loopEvents()) {

    double edep = 0;

    for (auto trk = dr.trackBegin();trk!=dr.trackEnd();++trk) {
      if (trk->pdgCode()==11/*electron*/) {
        for (auto seg = trk->segmentBegin(); seg!=trk->segmentEnd(); ++seg) {
          if (seg->volumeName()=="CountingGas")
            edep += seg->eDep();
        }
      }
    }
    printf("Energy depostion from e- in gas this event: %g keV\n",edep/Units::keV);
  }

  return 0;
}
```

providing meta-data access...

seemless looping over events in all requested inputfiles ...

and object oriented access to event data

Can of course be loaded from python as well!

If multiple input files are specified, Griff will protect you from yourself by checking that the files are compatible (i.e. their metadata implies consistent setup) ...

... unless you actually *want* to access different setups, in which case Griff will nicely inform you whenever the setup changes between events...

Same GRIFF analysis with filters and iterators rather than for-loops and if-statements


```
#include "GriffAnaUtils/All.hh"

int main(int argc,char** argv) {

  GriffDataReader dr(argc,argv);

  //Setup iterators and filters:
  GriffAnaUtils::SegmentIterator si(&dr);
  si.addFilter(new GriffAnaUtils::SegmentFilter_Volume("CountingGas"));
  si.addFilter(new GriffAnaUtils::SegmentFilter_EnergyDeposition(0.0));
  si.addFilter(new GriffAnaUtils::TrackFilter_PDGCode(11/*electron*/));

  //Loop over the events:
  while (dr.loopEvents()) {

    double edep = 0;
    while (auto seg = si.next())
      edep += seg->eDep();

    printf("Energy depostion from e- in gas this event: %g keV\n",edep/Units::keV);
  }

  return 0;
}
```

Setup iterators with associated filters
(convenient control of analysis “cuts”, just 1-line needs to be changed to add/remove a filter)

Usage in the event loop for very compact and readable code

purely optional of course, but highly recommended

… in particular useful as your analysis code grows and in danger of getting needlessly complicated and obscure

If needed, we can add more types of filters to the standard catalogue.
– or you can easily write your own

Same GRIFF analysis producing histograms rather than useless print-statements

REFERENCE

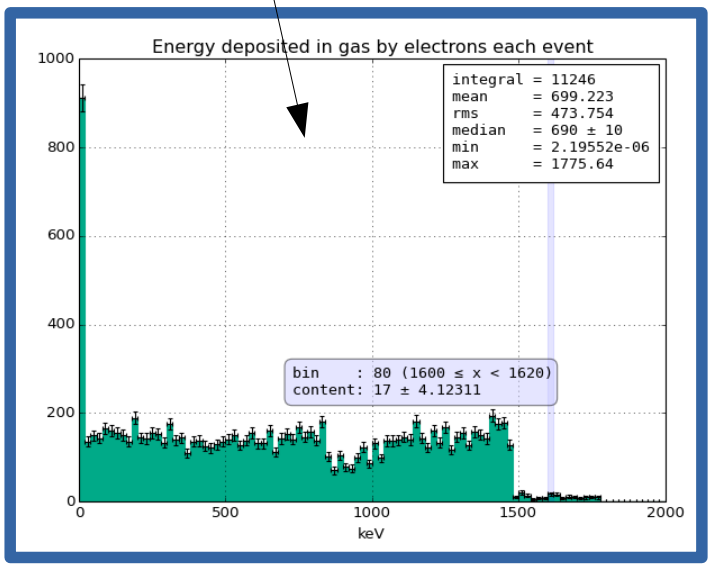
```
#include "GriffAnaUtils/All.hh"
#include "SimpleHists/HistCollection.hh"

int main(int argc,char** argv) {

  GriffDataReader dr(argc,argv);

  //Setup histograms
  SimpleHists::HistCollection hc;
  auto h_edep = hc.book1D("Energy deposited in gas by electrons each event",
                          100, 0.0, 2000, "h_edep");
  h_edep->setXLabel("keV");

  //Setup iterators and filters:
  GriffAnaUtils::SegmentIterator si(&dr);
  si.addFilter(new GriffAnaUtils::SegmentFilter_Volume("CountingGas"));
  si.addFilter(new GriffAnaUtils::SegmentFilter_EnergyDeposition(0.0));
  si.addFilter(new GriffAnaUtils::TrackFilter_PDGCode(11/*electron*/));

  //Loop over the events:
  while (dr.loopEvents()) {

    double edep = 0;
    while (auto seg = si.next())
      edep += seg->eDep();

    if (edep>0)
      h_edep->fill(edep/Units::keV);
  }

  //Save histograms
  hc.saveToFile("results.shist");

  return 0;
}
```

1) book

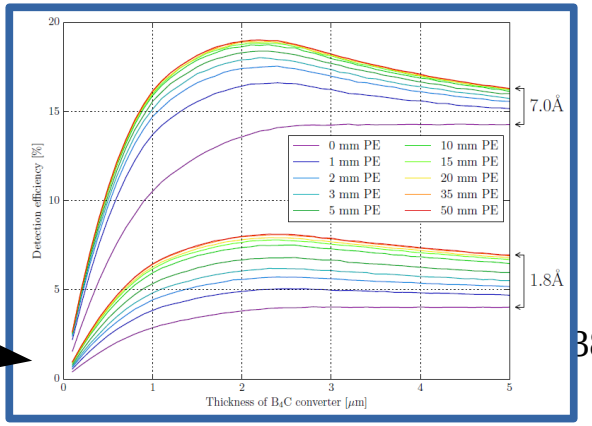
2) fill

3) write out

4) ess_simplehists_browse results.shist



And perhaps 5): Load the histograms in python and perform your final analysis there, if needed (perhaps combining results from different geometry setups, etc.)



Final analysis and plots with python scripts

REFERENCE

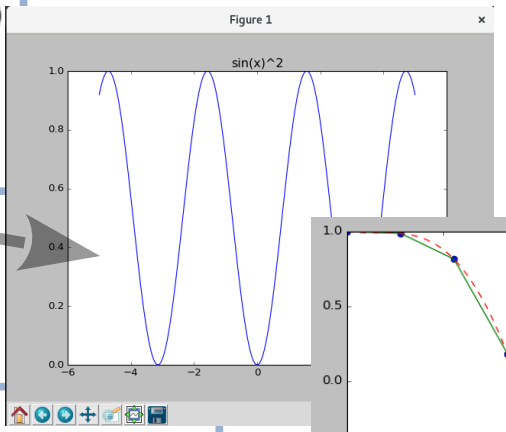
Using python (with matplotlib, numpy, scipy) is full-featured, freely available, and ties very nicely into everything else we do.

Simple "hello world" example:

```
#!/usr/bin/env python
from PyAna import *

x = np.linspace(-5.0,5.0,1000)
plt.plot(x,np.sin(x)**2)
plt.title('sin(x)^2')
plt.show()
```

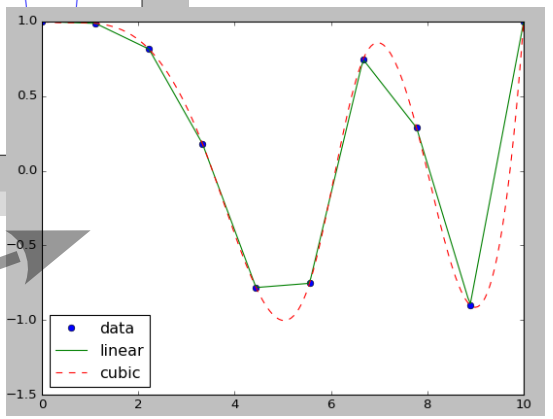
```
import os
import sys
import math
import matplotlib
import matplotlib.pyplot
import numpy
import scipy
import Core.Units
np = numpy
sp = scipy
mpl = matplotlib
plt = matplotlib.pyplot
Units = Core.Units
```

(+fixes)



Slightly more interesting example:

```
#!/usr/bin/env python

from PyAna import *
from scipy.interpolate import interp1d
#some fake data:
datax = np.linspace(0, 10, 10)
datay = np.cos(-datax**2/8.0)
#create linear and cubic interpolation functions based on the data:
f = interp1d(datax, datay)
f2 = interp1d(datax, datay, kind='cubic')
#plot:
x = np.linspace(0, 10, 200)
plt.plot(datax,datay,'o',x,f(x),'-', x, f2(x),'--')
plt.legend(['data', 'linear', 'cubic'], loc='best',numpoints=1)
plt.show()
```



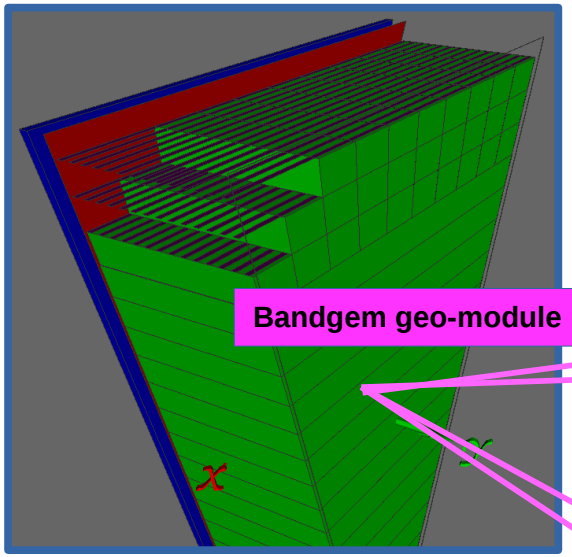
In reality, you will often open files produced in C++ analysis and just do final plotting (.shist, .griff, .txt)

Advanced material

For mature or would-be-mature projects

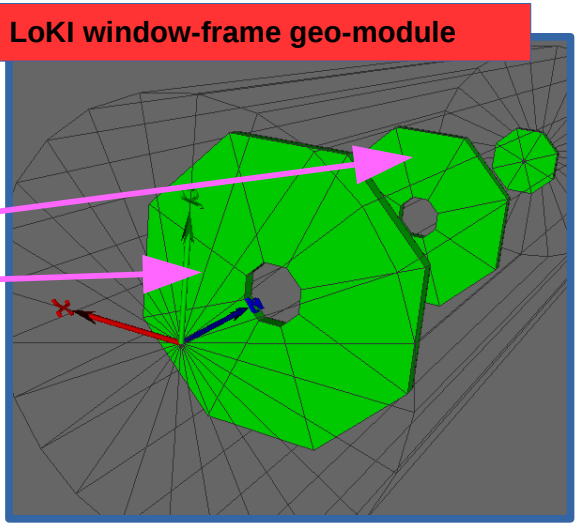
How to manage larger projects: Using geometry modules inside each other

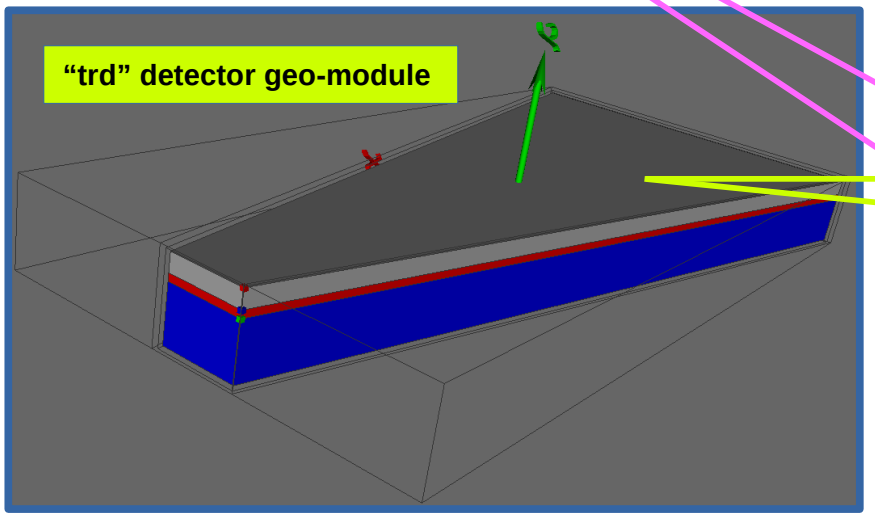
We now have a clear mechanism for how to do this!
(integral to this is our system for handling geo-parameters)



Bandgem geo-module

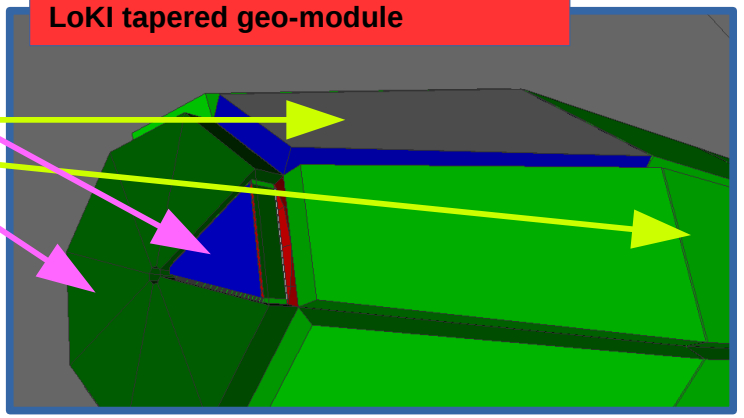
LoKI window-frame geo-module





"trd" detector geo-module

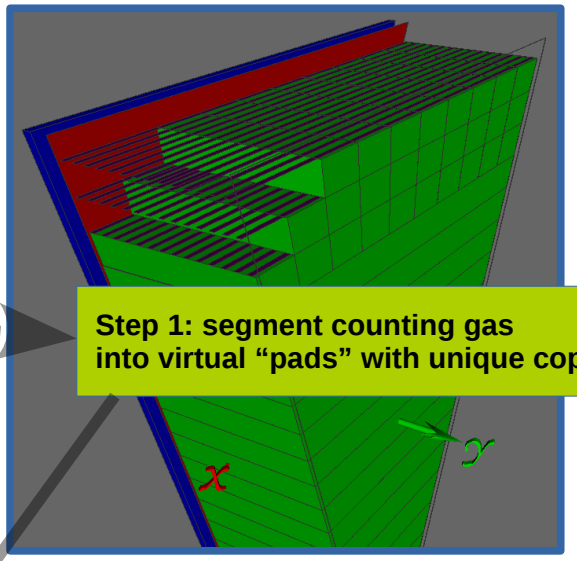
LoKI tapered geo-module



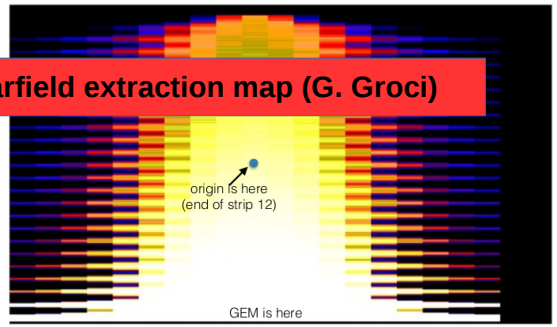
Project specific solutions (here bandgem)

Needed Geant4 sim:
- Integrate 2D extraction map, based on Garfield sim.
- Model readout segmentation (“pads”)
- Efficient high-stat simulation & analysis
- Data visualisation in pad-view

Had already: Geometry + Griff analysis.



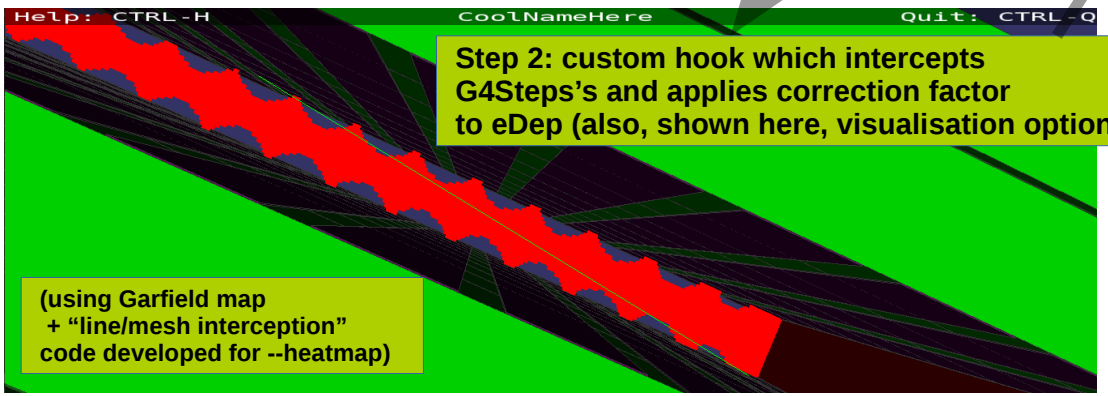
Step 1: segment counting gas into virtual “pads” with unique copyNbs.



Garfield extraction map (G. Groci)

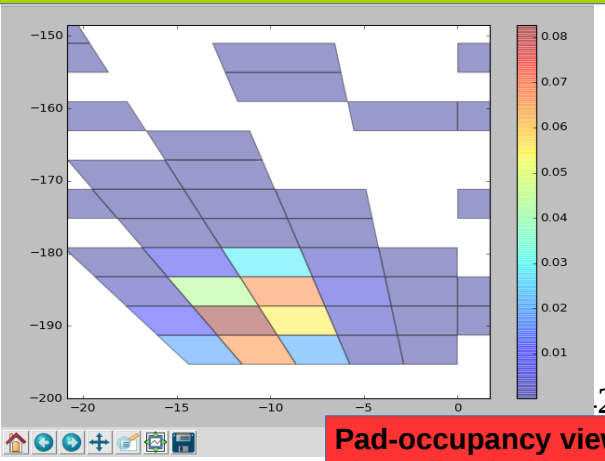
Step 3: custom simply binary data-format, writing edep-per-pad info for each event, directly from G4 job.

Step 4: Write small modules in C++ which extracts relevant info from data files and provides summary info to python-code for plotting. Write pad-viewer.



Step 2: custom hook which intercepts G4Steps’s and applies correction factor to eDep (also, shown here, visualisation option)

(using Garfield map + “line/mesh interception” code developed for --heatmap)

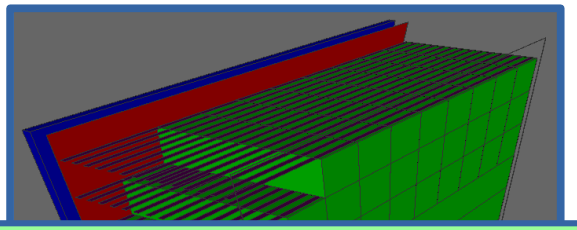


Pad-occupancy view, for a given Illumination.

Responsible: G. Albani + Milano group
Framework help from KK and TK

Project specific solutions (here bandgem)

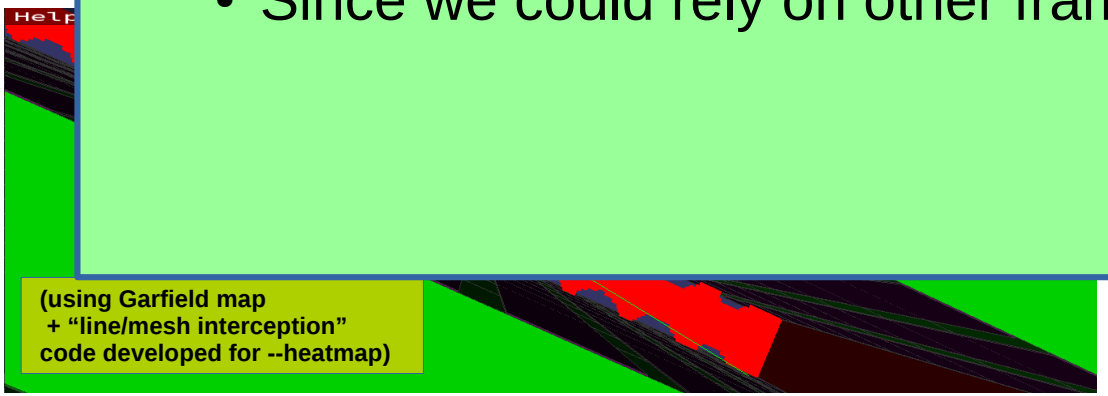
Needed Geant4 sim:
- Integrate 2D extraction map, based on Garfield sim.
- Model readout segmentation (“pads”)
-
-
H



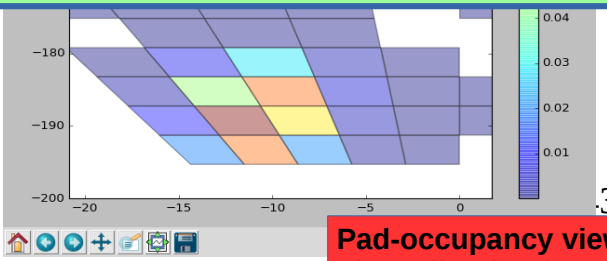
Most important point of this slide:

The code was developed in our framework, thus we could help!
- Since we could easily inspect, run, understand & modify it
- Since we could rely on other framework features for solutions

Garfiel



(using Garfield map
+ “line/mesh interception”
code developed for --heatmap)



Pad-occupancy view, for a given Illumination.

Responsible: G. Albani + Milano group
Framework help from KK and TK

Needing more computing power?

https://ess-ics.atlassian.net/wiki/display/DG/How+to+work+at+the+DMSC

- First of all, consider & ask if there is perhaps some obvious speedup you could implement via code-changes…
- If not, then you probably need to use the cluster @ DMSC !
- Unfortunately this means:
  - this ***will*** require you to be (even more) comfortable with the terminal and the ssh / scp commands.
- On the plus-side:
  - we provide utilities in dgcode for dealing with the batch system
  - we are here to help
  - the DG will get ~40TB of dedicated backed-up storage @ DMSC this year. This is intended for testbeam-data etc., so might as well learn how to get to it :-)

Parameter scanning

At some point you are going to want to make a plot of some analysis result as a function of one or more simulation parameter (geo/gen parameters, physics list, ...)

First of all, you better hope that all of the parameters in question can be modified from the commandline (you didn't hardcode too much, did you?)

Next, you are facing a potential book-keeping nightmare... keeping track of which results correspond to what parameters.

Fortunately, you are in luck!
(what a surprise after that intro!)

Assuming you are more or less following our standard simulation project setup, you simply first define which parameters you want to assume which values for which plots:

And then you have a script which can launch all of these jobs for you:

`ess_tricorder_scan`

```
#!/usr/bin/env python

from ScanUtils.ScanLauncher import ScanLauncher,ParameterGroup
from numpy import linspace

#Global setup for all scan jobs:
scan = ScanLauncher("ess_skeletonsp_simanachain",autoseed=True)
scan.add_global_parameter("rundir")
scan.add_global_parameter("--cleanup")
scan.add_global_parameter("--nevts",10000000)
scan.add_global_parameter('--physlist','QGSP_BIC_HP')
scan.add_global_parameter('material_lab','IdealGas:formula=CO2')

#scan jobs to investigate sample size effect:
plot1 = ParameterGroup()
plot1.add('sample_radius_mm', linspace(1.0, 20.0, 20) )
plot1.add('neutron_wavelength_aangstrom', [1.8, 2.2, 2.5])
scan.add_job(plot1,'plot1')

#scan jobs to investigate neutron wavelength effect:
plot2 = ParameterGroup()
plot2.add('sample_radius_mm', [5.0, 20])
plot2.add('neutron_wavelength_aangstrom', linspace(0.8, 10.0, 20) )
scan.add_job(plot2,'plot2')

scan.go()
```

TriCorder/scripts/scan

Voila, a script which you can use to launch your many jobs, either locally or on the DMSC cluster (from compile.esss.dk)

Pro-tip 1: reduce nevts/job to 1 and launch the scan locally first and catch any Configuration errors
(waiting 2 days in a cluster queue just to then have all jobs failing due to a misspelled variable name might ruin your mood)

```
(thki@tklenovo2014 TriCorder)> ess_tricorder_scan -h
Usage: ess_tricorder_scan [options]

Scan script which is used to launch the ess_tricorder_simanachain application
with various parameters. Note that it will invoke "dgbuild --install" before
launching jobs, in order to ensure build consistency and to prevent future
developments from interfering with running jobs

Options:
  -h, --help            show this help message and exit

  Main options:
    -s, --show          Show which jobs would be launched by --launch option
    --launch            Actually launch jobs

  Controlling launch aspects:
    -q QUEUE, --queue=QUEUE
                        Queue in which to launch jobs. Valid options are
                        "local", "dmsc:express", "dmsc:long", "dmsc:verylong"
    -d DIR, --dir=DIR   Non-existing or empty directory which will hold run-
                        dirs of scan jobs

  Options for DMSC queues only:
    -e EMAIL, --email=EMAIL
                        Email to alert upon job failures. Will read attempt to
                        read $EMAIL env var if not supplied.
    -r, --resubmit      Enable to attempt to continue an earlier failed
                        submission attempt. Will not reinstall dgcode.

  Options for local queue only:
    -j N                For "local" queue only, this sets the number of jobs
                        to run in parallel
    --halt-on-error     For "local" queue only, this option prevents further
                        jobs from being launched if any of them halts with an
                        error.
```

```
$>  ess_tricorder_scan -s
```

Show the job list

Launch locally

```
$>  ess_tricorder_scan -qlocal -d scanrundir/ -j4 --launch
```

Launch at DMSC

```
$>  ess_tricorder_scan -qdmsc:long -d scanrundir/ --launch
```

Pro-tip 2: You can also use this infrastructure to launch the same sim. setup over many cluster jobs (with different random seeds), and merge the results

Parameter scanning – analysing results

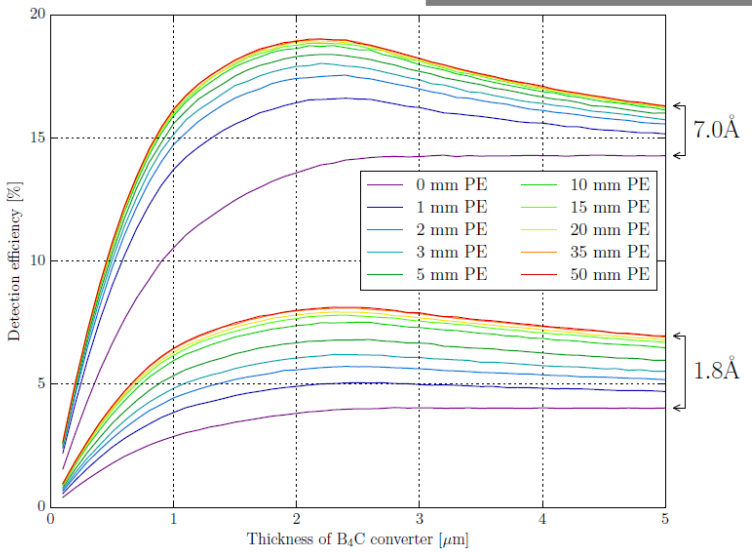
After scan is complete, you simply point our ScanLoader at the directory where you ran it (if on the cluster, you obviously first copy them down to your laptop)

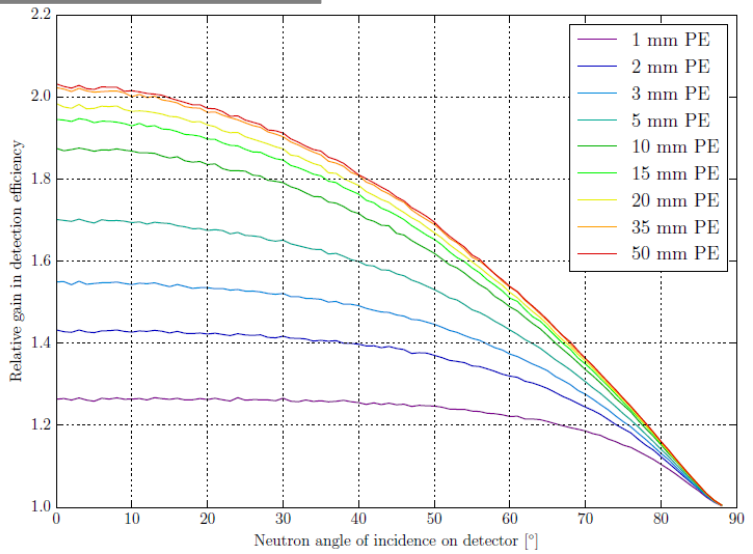
```
#!/usr/bin/env python

import ScanUtils.ScanLoader
from PyAna import *

jobs=ScanUtils.ScanLoader.get_scan_jobs(sys.argv[1],dict_by_label=True)

#Time to make some plots!! (beyond the scope of this tutorial...)
#Alas, I am guessing we are well out of time by now...
#But just to show you what info is available:
for j in jobs['plot1']:
    h_edep = j.hist('h_edep')                       #histograms for that job are available
    sample_size = j.setup().geo().sample_radius_mm #and metadata as well...
```

But getting a bit out of scope now for a 2-hour tutorial, so will simply show some example resulting plots on the next slide :-)

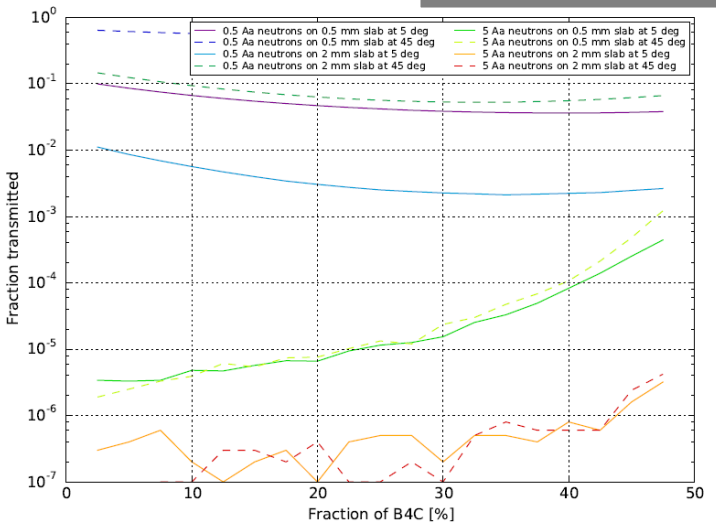
Parameter scanning – examples of results

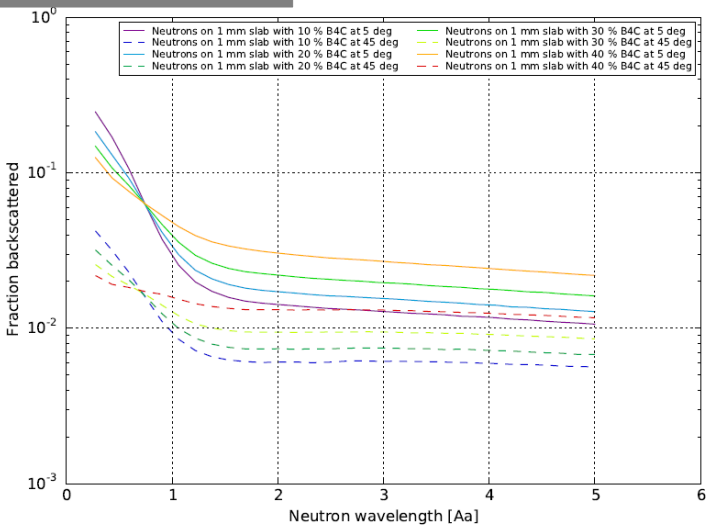
Study of detector efficiency dependency on various geometrical parameters





Study of how fractions of ingredients in a material mixture leads to different shielding properties





Fin

(almost)

Please remember: Wiki & DGSW

- Wiki has a lot of documentation, please make use of it!
  - https://ess-ics.atlassian.net/wiki/display/DGPrivate
  - https://ess-ics.atlassian.net/wiki/display/DG
  - https://ess-ics.atlassian.net/wiki/display/DG/Computing
- Please pose questions (even “stupid” ones), support requests, etc. by opening issues in the DGSW Jira project, rather than email.
  - https://ess-ics.atlassian.net/projects/DGSW

Fin

(for real)